ザ・C++

戸川隼人 著

ザ・C++

《CONTENTS》

NSライブラリ
RTRAN
SIC
CAL
BOL
O C
IC/98
・OS
・UNIX

AWAGOT OTAYAH

```
// example A.1
#include <iostream.h>

main ( )
 {
   int        n=753;
   float      h=3.1416
   double     k=123.4567;
   cout<<"&n="<<int (&n) <<¥n' ;
   cout<<"&h="<<int (&h) <<¥n' ;
```

サイエンス社

〈NSライブラリ〉

■計算機科学の急速な細分化・専門化の中で，その精髄をできるだけはやく，的確に把握し身につけたいとする人たちの真摯な要請に応えるべく，各分野の

必要にして十分（Necessary & Sufficient）

な内容を厳選し，わかり易く解説したのが，本ライブラリであります．

① **ザ・FORTRAN77**
戸川隼人著　本体1408円

② **ザ・BASIC**
戸川隼人著　本体1700円

③ **ザ・PASCAL**
戸川隼人著　本体1700円

④ **ザ・C［第2版］**
—ANSI C準拠—
戸川隼人著　本体1750円

⑤ **ザ・C++**
戸川隼人著　本体1900円

⑥ **ザ・数値計算リテラシ**
戸川隼人著　本体1480円

⑦ **ザ・TURBO C**
戸川・平島共著　本体1806円

⑧ **ザ・BASIC/98**
戸川隼人著　本体1750円

⑩ **ザ・UNIX**
戸川隼人著　本体1700円

⑫ **ザ・Fortran 90/95**
戸川隼人著　本体1750円

⑬ **ザ・Visual Basic**
戸川隼人著　本体1850円

⑭ **ザ・Java 2**
—対話的に動くホームページの作成技術—
戸川隼人著　本体1800円

⑮ **ザ・Linux**
戸川隼人著　本体1600円

※表示価格は全て税抜きです．

ザ・C++

戸川隼人 著

ザ・C++

《CONTENTS》

NSライブラリ

RTRAN
SIC
CAL
BOL
0 C
IC/98
・OS
・UNIX

AWAGOT OTAYAH

```
// example A. 1
#include <iostream. h>

main ( )
 {
   int        n=753;
   float      h=3. 1416
   double     k=123. 4567:
   cout<<"&n="<<int (&n) <<¥n' ;
   cout<<"&h="<<int (&h) <<¥n' ;
```

サイエンス社

ま え が き

この本は「Cを知らない人」のためのC++入門書です．コンピュータに関する予備知識があまりなくても読めるように，初歩的な説明も入れて，できるだけやさしく書いたつもりです．

C++はCを拡張した言語なので，まずCを勉強し，それをマスターしてからC++を勉強するのが当然と思われているようです．また，現在Cを使っている人がC++に転向する場合が多いので，C++の解説書の多くはCの知識を前提として書かれています．

しかし，はじめて勉強するのなら，先にCを勉強するより，直接C++を勉強する方がよいと思います．C++はCよりも機能が増えているので，むずかしいところもありますが，基本的な使い方に関してはC++の方がずっとやさしくて覚えやすいのです．特に学校教育の場合，Cで教えるよりもC++で教える方が教えやすく，落ちこぼれも少なくなると思います．

最近は学校教育で（BASICやFORTRANを教えないで）いきなりCを教える所が多くなってきました．本書はそのためのテキストで，C++の基本的な使い方を中心に説明してあります．これで要領がわかったら，上のレベルの本に進んで下さい．

1993年11月18日

H. Togawa

目　　次

4 配列と文字データの扱い方

5 関数の書き方と使い方

6 構造体とクラス

付録A ポインタ

1

操作法の要点

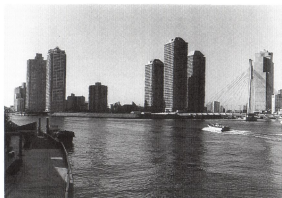
この章では，最初に

C++とは何か

という簡単な紹介をしたあと，

エディタの使い方

コンパイルの方法

など，C++のプログラムをコンピュータに入力し実行するための，操作法の要点を説明します．

1.1　C++とは何か

【初心者のための解説】 C++は新しいプログラミング言語（プログラムを記述するための言語）の一種です．

プログラムというのは「コンピュータに処理の詳細を指示する手順書」で，演劇におけるシナリオや音楽における楽譜のようなものです．私たちが計算の手順などをプログラムとして書いて，コンピュータに読み込ませると，機械はそれを解読し，その指示に従って自動的に動いてくれます．

プログラムは，私たちが「やってもらいたいこと」をコンピュータに伝えるための手段ですから，日本語で書いてコンピュータがそれを理解してくれればいちばん便利なのですが，人間が使っている言語は非常に複雑で，語彙が多く，表現も多種多様であり，

　　自動的な解読が困難
　　あいまいな点が残る
　　用途を限定すれば記号で書く方が簡単*

といった問題点があるので，かわりに人工言語を作り，

　　なるべく自然言語に近く
　　単純で解読し易い

ようにして，それによってプログラムを書くことになっています．これがプログラミング言語です．

---

* たとえば「aとbを加えて，その結果にcを掛けて，xに代入しなさい」と書くよりも，`x=(a+b)*c` と書く方がずっと簡単です．

現在，広く使われているプログラミング言語には

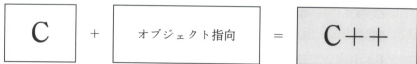
FORTRAN	数値計算の手順を表すのに便利な言語
COBOL	事務処理の手順を表すのに便利な言語
LISP	記号処理の手順を表すのに便利な言語
BASIC	初心者に覚え易い汎用言語
C	高度な処理を記述するのに適した言語

などがあります（右に付記した寸評は，十分な説明にはなっていませんが，まあ一応の常識です）．

この内，近年，Cを使う人がだんだん多くなってきています．その理由は，プログラムが高度化し，FORTRANやCOBOLでは表現しにくい処理が増えているためだと思います．

C++はCの文法を拡張し，もっと便利に，使い易くした新しい言語です．基本的な文法はCと全く同じなので，現在Cを使っている人は追加機能を勉強するだけで済み，これまでにCで書いたプログラムをそのまま利用することもできます．

機能が増えているので「勉強しなければならないこと」は当然多くなっていますが，ある面では易しくなっています．たとえば，入出力の書き方は非常に簡単になりました．また，これまでCを勉強する上での「最大の難所」とされてきたのがポインタで，Cのプログラムは各所にこれを使わないと書けなかったのですが，C++ではポインタの不自然な利用をしなくて済むようになりました．そういうわけで，C++は初心者にも腕利きのベテランにも適している言語であり，今後ますます広く使われるようになるでしょう．

【上級者のための解説】 C++は，Cを「オブジェクト指向」という方向に拡張した，新しい言語です．

オブジェクト指向というのは，プログラムの書き方を(これまでよりもっと)人間の常識に近づけ，現実の世界における物(オブジェクト)をありのままに表現できるようにしよう，という主張で，これを実現するために「クラス」という新しい概念が導入されています．従来のプログラミング言語（たとえばFORTRAN）には，

●互いに関係の深い変数を「まとめて扱う」ことができない

という弱点がありました．たとえば，分数を扱う場合，分母と分子を全く別の変数として（たとえば変数名**BUNSI**と**BUNBO**で表して）扱わなければなりませんでした．たくさんの分数を使って複雑な計算をする場合には，これでは非常に不便です．

そこでCでは「構造体」という書き方が導入されています．構造体というのは，要するに「利用者がデータ構造を定義し，複数個の変数をまとめて扱うことができるようにしたもの」で，これを使えば，たとえば上の例の場合，分数を表す構造体を定義して，

分母と分子をまとめて（一つの変数名で）扱う

ことができます（必要に応じて，分母だけ，分子だけを取り出したり，代入したりすることもできます）．しかし，構造体に関しては

●文法上の制約が多くて使いにくい

という悩みがありました．

クラスというのは，Cの文法における構造体の概念を拡張したものですが，これまであった問題点の大部分が解消し，強力な新機能が付加されて，非常に使い易いものになっています．

特に，構造体の間の演算その他の処理を表現するのに，演算子（+-*/=などの記号）の意味を利用者が自由に定義して使用できる点は便利で，プログラムがたいへん書き易く，かつ読み易くなりました．たとえば，先ほどの例と同様に分数を扱う場合，分数というクラスを作り，

　　+という記号が来たら，こういう計算をして下さい．
　　-という記号が来たら，こういう計算をして下さい．
　　*という記号が来たら，こういう計算をして下さい．
　　・・・・・・・・・・・・・・・

といった要領で演算子の意味を定義しておけば，

　　変数名a, b, cは分数を表す

と宣言して

```
c=a+b;
```

というような代入文を書くことができます．

従来の方式ですと，a, b, cを引数として「分数の和」のサブルーティン（あるいは手続き，関数など）を呼び出す必要があったわけですが，C++は上記のように普通の式の形で書けるのですから，ずっと便利ですね．

1.2 操作法の要点

文法の説明に入る前に（早くコンピュータで実習したい，という人が多いでしょうから）C++のプログラムを実行するための操作法を簡単に説明しておきます．一般に，プログラムを実行するには次のような作業が必要です．

1） プログラムを入力する

まず，キーボードを使って，プログラムを所定の形式でコンピュータに入れてやる必要があります．紙に書いたプログラムをコンピュータが読んでくれるといいのですが，現在のところ，ここはまだ手作業です．

プログラムの入力には「エディタ」というツールを使います．これには簡単なものから高機能で複雑なものまで，いろいろな種類がありますが，要するにワープロのようなソフトで，

キーボードから文字を入れる
必要に応じて修正を行なう
結果を標準的な形式でファイルに書き込む

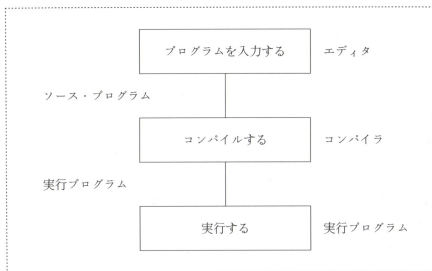
といった機能をもっています．

2） コンパイルする

前段1）で入力されたプログラムは，文字列の形式になっているので，そのままでは実行ができません．コンピュータで実行するためには，これを機械語に翻訳（コンパイル）してやる必要があります．

それを行なうのが「コンパイラ」というソフトです．C++のプログラムをコンパイルするにはC++コンパイラを使います．

3）実行する

コンパイルされた結果（機械語のプログラム）は「実行ファイル」という形式で記憶装置に入ります．それを呼び出して実行を開始するには，そのための操作が必要です．

具体的な操作法は使用機種やOS（オペレーティング・システム）によって異なります．操作法には大別して

●コマンド（操作指令）をキーボードから打ち込んで操作する

●マウスでアイコン（絵記号）を指示して操作する

の二通りがあります．両者の代表的なケースについて以下で簡単に説明しますが，詳しくは個々のソフトのマニュアルや解説書で研究して下さい．

1.3 TurboC++ for WINDOWS の場合

操作法が最も簡単なのは，MS-WINDOWS の上で TurboC++ を使う方式でしょう．

1）用意するもの*

この方式は PC-9801, EPSON, IBM 互換機（DOS/V マシン）の上で使用することができます．ハードディスクと 5MB 以上（できれば 8MB 以上）の RAM が必要です．ソフトとしては

MS-WINDOWS　および

TurboC++ for WINDOWS

が必要です．エディタは TurboC++ に含まれていますので，特に用意する必要はありません．

2）インストール*

ソフトを新しく購入した場合には，まずハードディスクに転送し，初期設定をしなければなりません．最近はこの種の組み込み操作（インストール）を自動的に実行してくれるようになりましたので，

所定のディスクを挿入する

所定の操作指令を入力する

画面に表示された指示に従って操作する

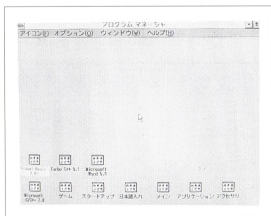
といった要領で行なうことができます．むずかしくはありませんが，間違えるとやっかいなことになりますので，細心の注意を払って進めて下さい．なお，WINDOWS のインストールには，かなり時間がかかります．

* 学校の実習などでは，既に準備ができていますから，この操作は必要ありません．次の 3) から始めて下さい．

3） TurboC++の起動

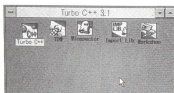
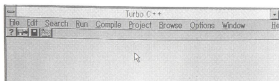
パソコンの電源スイッチを入れ，しばらく待っていると，まずMS-DOSが起動され，続いてWINDOWSが起動されます．もしもWINDOWSが自動的に起動されず，コマンド・モード（画面の左端に＞印が表示されている状態）になっている場合は，

WIN ↵

と打鍵してWINDOWSを起動します．

WINDOWSが立ち上がるとウィンドウ（窓）が現れます．アイコン（絵記号）がたくさん並んでいますね．その中からTurboC++のアイコンをさがし*，マウス・カーソルを合わせ，ダブルクリック（左上のボタンを2回たたく）します．

---

* 現在開いているウィンドウの中にTurboC++のアイコンがみつからない場合は，他のウィンドウを探さなければなりません．その説明は長くなるので省略します．WINDOWSの解説書を見て下さい．

どんな画面になったでしょうか？　下の図のように再びアイコンがたくさん並んだウィンドウが現れた場合は，その中の TurboC++（きっとある筈です）にマウスカーソルを合わせてダブルクリックして下さい．

そうすると，こんな画面になるでしょう．

この画面左上の **File** という所にマウス・カーソルを合わせてクリックするとメニューが現れます．その中（いちばん上）の **New** を選択して下さい（マウス・カーソルを合わせてクリックする）．新しい白紙のウィンドウが現れましたね．これで新規のプログラムの入力の準備が完了しました．

4）プログラムの入力

左記のようにNewを実行すると，自動的にエディタのモード（プログラムをキーボードから入力する状態）に入り，白紙の画面が出ます．

試しにABC… などとキーボードの鍵盤を押してみると，画面に表示され，ワープロと同じように使えることがわかるでしょう．それでは，36ページにある「ウィッキーさん」のプログラムを入れてみて下さい．

【初心者のための注意】

◆ 最初に，キーボードの状態が「英数字，小文字，半角」になっていることを確認して下さい．それには，キーボードからusoと入力してみるのがいいと思います．これで画面に

uso　　と表示されればOK

USO　　と表示されたら[CAPS]または[CAPS LOCK]というキーを押して「キャップス・ロック」状態（常に大文字に変換する状態）を解除します．

ﾅﾄﾗ　　と表示されたら[カナ]というキーを押して「カナ文字入力状態」を解除します（IBMキーボードの場合は[英数]というキーを押します）．

うそ　　または，ウソ，uso，USOなどと表示されたら，全角入力モードを解除する操作を行なう必要があります．操作法は機種と漢字入力システムによって違うので一概には言えませんが，PC-9801の場合は[CTRL]キーを押しながら[XFER]キーを押します．

◆ JIS キーボードの場合，記号 # ( ) " はキーボードの上の方（数字キーと共通），記号 <> は右下，記号 {} ; は上から2段目，3段目の右寄りにあります．記号の多くはキーの「上段」に刻印されていますので，SHIFT キーを押しながら使います．

（例） # を入れるには SHIFT を押しながら3のキーを押す．

コロン（:）とセミコロン（;）はよく似ていますので（特に目の悪い人は）注意して下さい．{} と ( ) も紛らわしいので，位置で覚えて下さい．

数字キーの上にあるのが丸カッコ

改行キーの横にあるのが波カッコ

です．記号 − に似た刻印がいくつもありますが，

数字 0 の右にあるのがマイナス

その二つ右（数字の段の右端）にあるのはカナ文字の長音記号

右下隅にあるのはアンダーライン（下線）

です．なお，

スペース（空白）は，いちばん下にある横長のキー

改行（次の行の先頭に移る）は右端にある ⏎ 印キー

で入力します（改行を enter, return と表している機種もあります）．

◆ 間違えたら

マウス・カーソルを誤りの箇所の右端に合わせる

BS キー（backspace，後退）を押して削除する

正しい文字を打鍵する

という要領で修正します．

◆ C や C++ では大文字と小文字が「別の文字」として区別して扱われます．FORTRAN や BASIC を使っていた人は注意して下さい．

【普通のエディタを知っている人のための解説】 TurboC++ for WINDOWS に組み込まれているエディタの基本的な操作法は，マッキントッシュや WINDOWS のソフトの標準的な操作法とだいたい同じと考えてよいでしょう．すなわち，エディタへの指示は，画面の上部に表示されているボタン

`File` `Edit` `Search` …

の中の一つをマウスで選択し，プルダウンしてメニューを開き，やりたいことをメニューの中から選んでクリックする，という方式で行ないます．

`File` には `New`, `Open`, `Save`, `Print`, `Exit`
`Edit` には `Undo`, `Redo`, `Cut`, `Copy`, `Paste`, `Clear`
`Search` には `Find`, `Replace`, `Search again`

などの機能が並んでいます*.

削除，複写，移動などの範囲の指定はマウスで行ないます．すなわち，対象とする範囲の先頭の文字にマウス・カーソルを合わせて左ボタンを押し，そのまま（ボタンを放さないで）対象範囲をなぞり，最後の文字まで移動させて，そこでボタンを放します．こうして先に範囲を指定し，それから `Cut`, `Copy` などを指示します．

複写や移動の目的位置の指定はマウスで指示します．すなわち，マウス・カーソルを「挿入したい場所」に合わせ，`Edit` メニューの中の `Paste` をクリックします．`cut` または `Copy` で取り込んだ内容はバッファの中に残っていますので，何度でも `Paste` を行なうことができます*.

* `Cut`（カット）は「切り取る」，`Paste`（ペースト）は「貼りつける」という意味．

5） コンパイル*

プログラムを最後まで入力したら，ウィンドウの上端に表示されている

Compile

というメニューを選び（左ボタンを押す），そこで表示されるメニューの中から，いちばん上の

Compile

をクリックします．そうするとコンパイルが開始され，もし重大なエラーが無ければ

Status:　Success

という表示が出るでしょう（成功！という意味です）．

誤りが発見されると，誤りの種類や位置に関する情報が表示されます．そうしたらプログラムを修正して，もう一度コンパイルをします．

6） 実行

コンパイルが完了したら「了解」という所をクリックしてコンパイラのウィンドウを閉じ，TurboC++のウィンドウの上端に表示されている

Run

をクリックし，表示されるメニューの中の「Run」を選択（左ボタンを押す）します．ここで最初に表示されるのは実行の準備段階（ローディングなど）を監視するウィンドウで，「了解」をクリックしてそのウィンドウを閉じると，「実行ウィンドウ」が現れます．今回の場合ですと，ここに

Have a nice day!

が表示されるはずです．うまくいきましたか？

---

* これをしないで直接6）実行の操作をしても自動的にコンパイルの処理がなされ，成功すれば実行に移ります．

7） 終了

結果を見たら，そのウィンドウの左上隅にある **−** 印をクリックしてウィンドウを閉じます．

次に **File** のメニューを開いて

Save as …

を選択してファイルをセーブ（ディスクに格納）します．

続けて次の仕事（新しいプログラムの入力など）に移るのであれば，**File** メニューの中の **New** とか **Open** を選択するわけですが，TurboC++ を終了するのであれば，同じ **File** メニューの中にある **Exit** を選択します．

それからプログラム・マネージャの「アイコン」というメニューの中から「WINDOWS の終了」を選択して終了させて，ディスクが静かになったら，念のために STOP キーを何度か押してから電源を切ります．

【注意】 ウィンドウがたくさん表示されている状態で電源を切ると，あとでシステムが混乱して動かなくなることがあります．必ず正規の終了手続きを済ませてから電源を切って下さい．もし操作法がわからなくなったら，電源を切らないで帰って（あるいは眠るとか，外出するとかして）下さい．

【TurboC++ の操作法のマトメ】 TurboC++ for WINDOWS は

エディタの呼出し

コンパイラの呼出し

実行開始の指令

が全部簡単なマウス操作によるメニュー選択できるので，たいへん便利です．

1.4　コマンド操作による方式の場合

1.4.1　概説

前節で説明したTurboC++ for WINDOWSは，大部分の操作がマウスによるメニュー選択だけで済みました．メニュー選択方式は「暗記しなければならないこと」が少なく，初心者でも安心して使えます．しかし普通はキーボードからMS-DOSのコマンドを用いて操作するので，

エディタの呼び出し方

コンパイラの起動法

実行開始の指令

などを覚えなければなりません．

前節で説明したTurboC++ for WINDOWSはエディタを内蔵していましたが，普通はC++コンパイラにエディタは含まれていないので，別途に用意する必要があります．

エディタにはいろいろな種類があり，続々と新製品が登場するので，一般的な説明はしにくいのですが，大きく分けて

ライン・エディタ

スクリーン・エディタ

の2種類があります．

ライン・エディタは

「第□行に～という処理をして下さい」

という形で操作を行なうエディタで，昔は広く使われていましたが，いろいろと不便な点が多いので，最近はあまり使われません．

一方，スクリーン・エディタは，表示画面の上でカーソルを動かして，自由に挿入，削除，書き換え等ができる便利なエディタです．特に最近のものは，非常に使い勝手が良くなっています．

以下では，両方式の代表的なソフトを取り上げて，基本的な操作法の要点をご紹介しましょう．

1.4.2 EDLIN の使い方

これはライン・エディタなので少々不便なのですが，MS-DOS の中に含まれているので（標準付属品のようなもの），

タダで使える

どこに行っても必ずある

という利点があります．

起動の方法

EDLIN ファイル名 ⏎

（例） EDLIN NICEDAY.CPP

これは「右に書いた名前のファイルの編集を開始して下さい」という命令だと思ってよいでしょう．

既にその名前のファイルがあれば，それが「編集用バッファ」に読み込まれます．もし，その名前のファイルがまだ無ければ，新規ファイルを作成することになります．

入力開始

I ⏎

I は insert の頭文字で「挿入する」という意味です．

プログラムやデータの入力

行の左端に

　　行番号:*

(例)　1:*

という印が表示されたら「入力OK」ということですのでプログラムやデータをキーボードから入れましょう.

入力の要領は11ページで説明した「TurboC++の場合」と基本的には同じですが，ライン・エディタは間違って入れた場合の修正がしにくいので慎重に入力して下さい．万一間違えた場合は,

- ◆ 修正すべき箇所が「現在入力している行」の中であれば，[BS]（後退）キーを何回も押して修正箇所にもどり，そこから全部入れ直します.
- ◆ 修正すべき箇所が「別の行」にあるときは，すぐに戻って修正することができないので，あとでまとめて修正するための要点を紙にメモして先に進みます.

挿入モードの終了

入力が終わったら

　　[CTRL] + [C]　([CTRL]キーを押しながら[C]のキーを押す)

によって「挿入モード」を脱出して「コマンド・モード」に戻ります.

表示して確認

入力したプログラムを表示して，内容を確認しましょう．1行だけを表示するには

　　行番号 P [↵]

(例)　12P [↵]

何行もまとめて表示するには

先頭行番号 , 終端行番号 P ↵

の形式で打鍵します．たとえば

3,7P ↵

と打鍵すれば第3行から第7行までが表示されます．プログラムの最初から最後まで全部表示するには

1,#P ↵

と打鍵します（#印は「最後の行」を表します）．行数が多くて1画面で表示できない場合は，1画面分ずつ止まって表示してくれます（続きを表示するか否かを聞いてきますのでYかNで答えます）．

挿入

行番号 I ↵

このあと入力した内容（何行でもよい）は，ここで指定した番号の行の直前に挿入されます．挿入モードの終了は，前記のとおり，

CTRL + C

です．

削除

行番号 D ↵

何行も一度に消したい場合は

先頭行番号 , 終端行番号 D ↵

エディタの終了

E ↵

結果はEDLINの引数として指定したファイルに書き込まれます．

【注意】 プログラムの修正作業をするとき「行番号は不変ではない」という点に注意して下さい．一般にライン・エディタの行番号は「先頭から数えて何行目」ということを指示するためのもので，

I 挿入

D 削除

などの操作をすると，その後の部分の行番号が影響を受けます．したがって，修正すべき箇所の行番号をメモしておいても，修正作業をする時点までに行番号が変わっているかもしれません．

そのような混乱を避けるためには，プログラムの最後の方から修正を始め，順に前の方の修正に進むのが賢明です．

また行番号を間違えて「修正する必要のない行」を削除してしまったら大変ですから，削除する前に必ず **L** または **P** コマンドで表示して，内容を確認するようにして下さい．

【便利な機能】 以上で説明した基本的なコマンド

I　D　L　P　E

を使って根気よく操作すれば，一応，どんなプログラムでも入力できます．

しかしこれだけだと，たった1字のミスを訂正するために1行全部を入れ直さなければなりません．キーボード操作に慣れていない初心者にとっては相当な苦痛ですね．この悩みを解決するために「置換」の方法を覚えましょう．

置換

これは「指定した行の中の旧文字列をすべて新文字列に書き換える」という機能を持ったコマンドで，

　行番号 R 旧文字列 [CTRL]＋[Z] 新文字列 [↵]

または

　先頭行番号 , 終端行番号 R 旧文字列 [CTRL]＋[Z] 新文字列 [↵]

の形で打鍵します．

(例)　第1行の

```
#include <iostream.h>
```

を誤って

```
#incrude <iostream.h>
```

と入力してしまった場合，これを修正するには

　1 R incrude [CTRL]＋[Z] include [↵]

あるいは，

　1 R cr [CTRL]＋[Z] cl [↵]

と打鍵します．もっと簡単に

　1 R r [CTRL]＋[Z] l [↵]

としてよさそうですが，そうすると iostream の r が l に書き換えられてしまいますので，注意して下さい．

EDLIN には，以上のほか，綴りの検索のためのコマンド（S）や，行をまとめて移動または複写するためのコマンド（M, C）があります．また「現在位置」を基準にして，その「何行前」「何行後」を指示することもできます．

1.4.3 FINAL の使い方

FINAL は初心者にも使いやすいスクリーン・エディタです．他のソフトの一部に含まれているエディタとは違って，独立したソフトですので，自分で購入しなければなりません（でも，実習室のパソコンには入っているかもしれません．一応，先生に聞いてみましょう）．コンパクトにできているので，ハードディスク無しでも使えます．

起動の方法

　　fe ファイル名 ↵

既存のファイル名を指定すると，そのファイルが読み込まれ画面に表示されて，ただちに修正を行なえる状態になります．

新しいファイル名を指定すると，白紙の画面が現れ，プログラムやデータを入力できる状態になります．

終了の方法

キーボード最上段にある f·1 というキーを押すと，終了方法を選択するためのメニューが表示されます．普通は，

　1． 現テキストのセーブ，編集終了

を選びます（そのメニューがハイライトされている状態で ↵ キーを押す）．
何か大きな失敗をして，編集を最初からやりなおしたい場合は，

　6． 全テキストの放棄，強制終了

を選びます（普通のように 1 を選ぶと，失敗したファイルがセーブされ，大切なもとのファイルが消されてしまいます）．

プログラムやデータの入力

初心者は，あまり高級な入力技法を使うよりも，キーボード操作を練習するつもりで，原稿（下書き）のとおり1字ずつ入れるのがよいと思います．一般的な注意は12ページに書いておきましたので参考にして下さい．

ミスの修正

一般に，スクリーン・エディタでは，画面上で自由にカーソルを動かして，誤りのある箇所に合わせ，

不要な文字を消す

追加すべき文字を入力する

という要領で修正を行なうことができます．

◆ **カーソルの移動**　初心者は矢印キー

[→][←][↑][↓]

だけで十分でしょう．遠くへ移動する場合でも，矢印キーを押し続ければ，かなり速く動いてくれます．

◆ **挿入**　普通は常に「挿入モード」になっていますので，特別な操作は要りません．もし「上書きモード」になっている場合，[INS]キーを押せば挿入モードに戻ります．

◆ **削除**　削除の操作には[DEL]キーまたは[BS]キーを使います．

[DEL]（削除）　カーソル位置の文字を削除

[BS]（後退）　カーソルの1字左の文字を削除

◆ **アンドゥ（削除した文字の復活）**

[CTRL]＋[B]

検索，置換

他のエディタやワープロと同様に，

検索　指定した文字列を含む行にカーソルを移す

置換　指定した文字列を別の文字列に書き換える

の処理が簡単にできます．この種の機能はメニュー6に含まれていますので，

[SHIFT]キーを押しながら[f·3]を押す

という操作によってメニュー6を表示し，

[↓][↑]キーで項目を選択して[↵]キーを押す

メニュー番号をキーボードから入れる

のどちらかの方法で機能を選び，あとは画面に表示されるメッセージにしたがって操作します．

◆ **検索**　メニュー6の中に

文字列の前方検索　([↓])

文字列の後方検索　([↑])

があります．そのどちらかを指定すると，検索文字列を問合わせてきますので，検索文字列を入れてリターン・キー　([↵])　を押します．

◆ **置換**　メニュー6の中に

文字列の全置換　(確認なし)

文字列の全置換　(確認あり)

文字列の全置換　(範囲指定)

があります．初心者はここで「確認あり」を指定しておくのが安全です．

そうすれば，該当箇所がみつかるたびに，

置換しますか？（y/n）

と聞いてきますので，予想外のトラブルを避けることができます．

［解説］置換は綴りの区切りに関係なく，たとえ文字列の一部であっても「探索文字列」に一致すれば変換してしまいますので，たとえば

in を **out** に書き換える

という処理を「確認なし」で実行すると，

int sin print include

などがみんな変換されてしまい，もとに戻すのに苦労するでしょう．

◆ **探索や置換の再実行**　メニュー6の中の

8．前回の探索置換の再実行

によって指令します．

移動，複写

エディタではワープロと同様に，既に入力されている文字列を他の場所に移動したりコピーしたりすることができます．行の一部分だけ（たとえば変数名や式など）を移動，複写することもできますし，何行もまとめて動かすこともできます．FINAL では，これを

まず，元の文字列の範囲を指定する

次に，機能を指定する

最後に，行先を指定する

という要領で操作します．

◆ **範囲の指定**　まず，移動・複写したい文字列の先頭（最初の文字）にカーソルを合わせて

[f·6]（画面下端のガイドには Sel と表示されている）

を押します．

それから，矢印キー（[→][←][↑][↓]）でカーソルを動かして，元の文字列の終端（最後の文字）の次の位置までカーソルを移動させます．カーソルを動かしていくと「指定された範囲」がハイライト（反転表示）されていきますので，確認して正確に範囲を指定できます．

◆ **機能の選択**　ここで，移動するか複写するかによって

[f·7]（画面下端のガイドには Cut と表示されている）

[f·8]（画面下端のガイドには Copy と表示されている）

のどちらかを押します．

[f·7] ならば元の文字列は削除されます（移動）．

[f·8] ならば元の文字列はそのまま残ります（複写）．

[f·7]を押すと，元の文字列が画面からパッと消えてしまい，ちょっと不安になるかもしれませんが，内容は「バッファ」という所に保存してありますので心配はいりません．

◆ **行先の指定**　最後に，カーソルを「挿入したい位置」に合わせて

[f·9]（画面下端のガイドには Paste と表示されている）

を押します．バッファに保存してあった内容は，カーソル位置を先頭にして挿入されます．この際，バッファの内容は消えずに残っていますので，必要に応じて何か所にでも複写を行なうことができます*.

---

* いわゆる「カット＆ペースト」という方式です．

1.4.4 VZ の使い方

VZ エディタはコンピュータ使いの達人に愛用されている高機能エディタです．何でもメニュー選択で操作できる FINAL と違って，機能がすべてキーボード操作に割り当てられているので，最初に覚えなければならないことが多いのが難点ですが，慣れれば非常に高速な操作が可能になります．

起動の方法

vz ファイル名 [↵]

既存のファイル名を指定すると，そのファイルが読み込まれ画面に表示されて，ただちに修正を行なえる状態になります．

一方，新しいファイル名を指定すると，

そのファイルは見つかりません．新規ファイルですか？

というような表示が出て，ここで [Y]（yes の意味）を打鍵すると白紙の画面が現れ，プログラムやデータを入力できる状態になります．

エディタの終了

[ESC] [Q]

[ESC]（エスケープ）キーを押し，それから[Q]のキーを押します．一太郎や CCT98 などと同じですね．結果は自動的にセーブされます．

プログラムやデータの入力

一般のスクリーン・エディタと同様に，画面上で自由にカーソルを動かし，文字キーを押せば，カーソルの位置に入力されます．誤りに気づいたら，カーソルを「修正すべき箇所」に合わせ，不要な文字を削除し，追加すべき文字を挿入する，という要領で修正します．

カーソルの移動

初心者は矢印キー

[→] [←] [↑] [↓]

およびスクロール・キー（[ROLL UP]，[ROLL DOWN]）だけを使っていれば十分です．しかしだんだん慣れてきて，もっと遠くまで迅速に移動させたくなったら，マニュアルを調べて

[CTRL]＋[←]　…行の先頭までスキップ

[CTRL]＋[→]　…行の最後までスキップ

[CTRL]＋[ROLL UP]　…ファイルの最初の行に戻る

[CTRL]＋[ROLL DOWN]　…ファイルの最後の行までスキップ

など，便利な機能を活用しましょう．

ブラインド・タッチ（鍵盤を見ないで打つテクニック）のできる方々には，矢印キーやスクロール・キーを使うよりも「[CTRL]キーを押しながら文字キーを押す」という方式をお奨めします．これは，キーボードの左中央にある

　　[E]　　　　　　　　　　　　[↑]

[S]　　[D]　を[CTRL]と同時に押すと　[←]　　[→]

　　[X]　　　　　　　　　　　　[↓]

の方向にカーソルが動く，というのが基本で，それに隣接するキーが，やはりカーソル移動の機能を持っており，たとえば，

[CTRL]＋[R]…一つ前（上）のページを表示して移動

[CTRL]＋[C]…一つ後（次）のページを表示して移動

といったぐあいです．

挿入

普通は常に「挿入モード」になっていますので，特別な操作は要りません．「上書きモード」になっている場合，[INS]キーを押せば挿入モードに戻ります．

削除

削除の操作には

[DEL]（削除）　…カーソルがある位置の文字を削除
[BS]（後退）　…カーソルの１字左の文字を削除

のほか，VZ では

[CTRL]＋[DEL]　…カーソル位置から右（行末まで）を削除
[CTRL]＋[BS]　…カーソル位置から左（行頭まで）を削除
[SHIFT]＋[DEL]　…カーソル位置から単語の右端までを削除
[SHIFT]＋[BS]　…カーソル位置から単語の左端までを削除

などの操作ができます．

アンドゥ（削除した文字の復活）

一太郎と同じ

[CTRL]＋[U]

です．

検索，置換

画面の下端を見ると，

ﾌｧｲﾙ　窓換　文換　窓割　記憶　検索　置換　ｶｯﾄ　ｲﾝｻｰﾄ　ﾌﾞﾛｯｸ

という表示が出ていますね．これはキーボード最上段にある

[f･1][f･2][f･3][f･4][f･5][f･6][f･7][f･8][f･9][f･10]

という刻印のあるキー（ファンクション・キー）に対応しています．

検索する場合は，画面の「検索」に対応する f·6 を押すと検索文字列を問い合わせて来ますので，あとは

検索文字列を入れてリターン・キー（↵）を押す

カーソル位置から順方向（画面下方）に探す場合は CTRL + C

カーソル位置から逆方向（画面上方）に探す場合は CTRL + R

という要領で操作します．

置換する場合は，画面の「置換」に対応する f·7 を押すと

検索文字列（変換すべき旧文字列）

置換文字列（変換して置き換えるべき新文字列）

範囲（カーソルより先か，カーソルより前か，全域か）

方式（全部一斉に置換するか，一つずつ確認するか）

を問い合わせて来ますので，画面に表示される指示にしたがって操作します．

移動，複写

カット&ペースト方式なので，

元の文字列の範囲*を指定する

機能を指定する

行先を指定する

という順序で操作します．

◆ **範囲*の指定** 先頭（最初の文字）にカーソルを合わせて

f·10 （画面下端のガイドにはﾌﾞﾛｯｸと表示されている）

を押します．それからカーソルを動かして，終端（最後の文字）まで移動させます．

* VZ の用語では，まとめて動かす範囲（区画）をブロックといいます．

◆ **機能の選択**　ここで，移動するか複写するかによって

[f·8]　（画面下端のガイドにはｶｯﾄと表示されている）

[SHIFT] + [f·8]　（画面下端のガイドにはｺﾋﾟｰと表示されている）

のどちらかを押します．

[f·8]　ならば元の文字列は削除されます（移動）．

[SHIFT] + [f·8]　ならば元の文字列はそのまま残ります（複写）．

◆ **行先の指定**　最後に，カーソルを「挿入したい位置」に合わせて

[f·9]　（画面下端のガイドにはｲﾝｻｰﾄと表示されている）

または

[SHIFT] + [f·9]　（画面下端のガイドにはﾍﾟｰｽﾄと表示されている）

を押します．

バッファに保存してあった内容は，カーソル位置を先頭にして挿入されます．ペースト（[SHIFT] + [f·9]）の場合には，バッファの内容は消えずに残っていますので，必要に応じて何か所にでも複写を行なうことができます．インサート（[f·9]）の場合には1回だけです．

【解説】　これだけならばFINALとあまり変わらないようですが，じつは次のような大きな違いがあります．

1)　範囲の指定をせずに（すなわち[f·10]を押さずに）

[f·8]　（ｶｯﾄ）

[SHIFT] + [f·8]　（ｺﾋﾟｰ）

の操作をすると「現在カーソルが指示している行」が対象になります．

2)　バッファはスタック（いくつも詰め込める方式）になっていて，複雑なカット＆ペースト操作が可能です．

1.5　コンパイルと実行

コマンド・モード

プログラムの入力が完了し，エディタを出ると，OS（オペレーティング・システム）の普通の状態に戻ります．MS-DOS ならば，ここで

DIR　　ファイル一覧表を表示する
TYPE　　ファイル内容を表示する
PRINT　プリンタで印刷する
COPY　　ファイルを複製する
REN　　ファイルの名前を変更する
DEL　　ファイルを削除する

などの操作を行なうことができます．COPY コマンドを使ってフロッピ・ディスクへのセーブ（保存，書込み）を行なうこともできます．

実行までに必要な操作

C++ のプログラムを実行するには，このあと，

コンパイルする
実行開始

の操作をしなければなりません．詳しくいうと，そのほか，

文法チェックを行なう
リンクする
ロードする

などの操作も必要なのですが，「コンパイル」と「実行開始」を指令すれば，リンクやロードの標準的な処理をたいてい一緒にやってくれますので，特別な扱いを希望するのでなければ，特に指令する必要はありません．

コンパイル（およびリンク）の指令

① マイクロソフトC++の場合*

CL ファイル名 ⏎

（例） CL NICEDAY.CPP

② Zortech社のC++の場合

ZTC ファイル名 ⏎

（例） ZTC NICEDAY.CPP

実行開始

コンパイルされた結果（ただちに実行できるプログラム）は，普通，

もとのプログラムのファイル名の語幹 **.EXE**

（例） **NICEDAY.EXE**

というファイル名になります．これを実行するには，その「**.EXE**」を省いた形（語幹）を打鍵して⏎キーを押します．

実行ファイル名の語幹 ⏎

（例） NICEDAY ⏎

【備考】 上の例では大文字を使っていますが，MS-DOSでは大文字と小文字を区別しないので，小文字で操作しても構いません．

* PWBを使用すればメニュー選択だけで操作できます．

うまく動かない場合

学校のパソコン教室における実習のように「本来ならばマトモに動くはずの環境」が整備されている場合には，動かない原因は

操作ミス

プログラムのミス

しか考えられませんから，入念に見直して，それでもわからなかったら友達や先生に聞きましょう．

一方，自宅のパソコンではじめてC++を使う場合には，初歩的なミスとして「メモリー不足」とか，コンパイラをインストールするときに操作ミスがあったとか，いろいろな可能性があります．

よくありそうな原因としては，パスとカレント・ディレクトリの指定の誤りが考えられます．詳しくはMS-DOSの参考書を見ていただきたいと思いますが，簡単にいえば，

パス（path）でコンパイラの所在を指示する

カレント・ディレクトリはプログラム・ファイルの場所に置く

とするのが原則です．念のため，

```
path ⏎
```

と打鍵して，コンパイラのディレクトリがその中に含まれているかどうかを調べ，また，

```
dir ⏎
```

でカレント・ディレクトリの内容を表示して，いま入力したプログラムのファイルが確かにあるかどうか確認して下さい．

2

数式の計算と入出力

この章では数式の計算と入出力の例を中心に，C++のプログラムの基本形を勉強します.

基本的な文法に関してはC++はCとだいたい共通ですので，既にCをよくご存知の方は簡単に眺めるだけで十分です．ただし，

入出力の書き方　`cin, cout`

注釈の書き方　`//`

などが使いやすい形になり，また，初期のCに無かった

拡張倍精度型　`long double`

などが追加され，数学関数ライブラリの扱い方が少し変わっていますので，そこだけ注意深く読んで下さい.

2.1 ワンポイントC++会話

プログラム例2.1——ウィッキーさん

画面に

Have a nice day!

と表示するプログラムは次のように書きます*.

```
// example 2.1
#include <iostream.h>

main( )
{
    cout << "Have a nice day! ¥n";
}
```

【解説】 普通，Cの簡単なプログラムは次のような形式で書きます*.

```
#include    <iostream.h>
main( )
{
    処理手続きの記述
}
```

【実行例】

```
Have a nice day!
```

* include の後の <> 内に書く「ヘッダーファイル名」は処理系によって少し違いがあります．もし iostream.h でエラーになったら stream.h を試み，それでもだめならマニュアルでよく調べて下さい．

注釈

最初に書いてある

```
// example 2.1
```

は注釈（コメント）です．これはプログラム作成者の覚え書きで，実行時には無視されます．

インクルード

その次の

```
#include <iostream.h>
```

は入出力の関数 **cin** や **cout** を使うための準備です．

文法解説

#印はプリプロセッサの記号　行の先頭に # 印を付ける書き方はプリプロセッサといって，実行前（正確にいえば「コンパイルの前」に処理すべき作業を指示します．よく使われるのは

```
#include    <ファイル名>
```

という形の書き方で決まり文句を読み込むのに使います．たとえば今の例のように

```
#include    <iostream.h>
```

と書いておくと，**iostream.h** という名前のファイルに書かれている内容が読み込まれ，その位置（**#include** が書いてあった所）に挿入されます．

C++を快適に使うためには，プログラミングの名人が作ってくれた便利な道具を組み込むとよいのですが，そのために

```
#include    <ファイル名>
```

の形が用いられます．

メイン・プログラム

メイン（主）プログラムというのは，起動したとき最初に実行されるプログラムで，ここからいろいろな副プログラムを呼び出して計算を進めていくのが普通です．

メイン・プログラムには必ず **main** という名前を付けます．C や C++ では，この **main** を文法上（形式的に）「関数」の一種として扱っています．平方根や三角関数の値を計算する普通の関数と違って，**main** には引数も関数値もありませんが，関数と同じ規則（文法）で書く決まりになっています．

記号 **{** および **}** は「始まり」と「終り」を表しています（PASCAL などにおける **begin**, **end** に相当）．C++ には「プログラムの終りを表す文」（たとえば FORTRAN の **END**）はありません．実行は，最初の **{** に対応する **}** に来れば終了します．

文の終りには ; 印をつける

プログラムは，文（statement）をいくつも（実行すべき順に）並べて記述します．一つ一つの文の末尾には必ずセミコロンを付けます*．

---

* FORTRAN や BASIC では「文は原則として 1 行で書き，行の終りが文の終り」という約束になっていますが，C や C++ や PASCAL では，

> 文の終りは「**;**」印で表す
> そのかわり一つの文が何行になってもよい
> また，1 行にいくつもの文を書いてもよい

という規則になっています．

出力

cout は出力をしてくれる「受付窓口」です．出力したいデータを受付窓口に提出すると cout がそれを受け取って画面に表示してくれるのです．

文字列を出力する方法*

```
cout << "出力すべき文字列" ;
```

変数の値を出力する方法

```
cout << 変数名;
```

¥印は特殊記号を表す

Have a nice day! の後に付いている **¥n** は改行の記号です．この記号を付けなくても，たいていは表示してくれますが，処理系によっては **¥n** を付けないと表示してくれない場合がありますので，文末には必ず **¥n** を書く習慣をつけましょう．

!!!

国産機では ¥n ですが，外国機では \n という記号を用います．

文字定数

文字列は2重引用符 " で囲みます．これを忘れると，変数名や命令語などと誤解されて，大量のエラーメッセージが出るでしょう．

---

* C で出力に使われている printf という関数は C++ でも，C と全く同様に使用できますが，cout の方が便利なので（書式指定をしないで済む），早く cout に慣れる方がよいでしょう．

2.2 整数の計算

プログラム例2.2——$i+j=k$

二つの整数 i, j を読み込み，

$$i+j=k$$

の計算をして表示するプログラムは次のように書きます．

```
// example 2.2
#include <iostream.h>

main( )
{
    int i,j,k;
    cout << "i=";
    cin >> i;          // i を読み込む
    cout << "j=";
    cin >> j;          // j を読み込む
    k=i+j;             // 計算
    cout << "k=" << k << "¥n"; // k を表示
}
```

整数型変数の宣言

メイン・プログラム本文の最初にある

```
int i,j,k;
```

は「変数 **i**, **j**, **k** は整数型である」という宣言です．整数型というのは，整数の計算をするための数値表現形式です．整数型は，扱うことのできる桁数が意外に少ないので注意して下さい（桁数は機種によって違います．普通は8桁ぐらいまで使えますが，5桁ぐらいしか扱えない機種もあります）．

【注意】 FORTRAN や BASIC と違って，C や C++ では必ず型宣言を書かなければいけません．また，変数の型宣言は，その変数を使用するより前に書かなければいけません*．

* C ではプログラムの冒頭に書きます．C++ では，その変数を最初に使用する直前に書くのが普通です．

入力促進メッセージ

7行目の

```
cout << "i=";
```

は入力促進メッセージ（プロンプト）を表示するためです．もっと丁寧に

```
cout << " i の値を入れて下さい ";
```

としてもよいでしょう．

データ入力の書き方

cin は読み込んだデータをもらう「受取り窓口」です．

```
cin >> i;
```

は受取った値を変数 i に入れることを表しています．

計算と代入

11行めの k=i+j; は「i+j の値を計算し変数 k に代入する」ということを表しています．他のプログラミング言語と同様に，C++でも

計算式は = の右側に書く

代入先は = の左側に書く

という規則になっています．

結果の出力

最後の cout … は，

k= という文字列，　変数 k の値，　改行の指示

を，この順に出力することを表しています．

【実行例】

```
i=2
j=3
k=5
```

プログラム例2.3——和差積商

二つの整数 i, j を読み込み

$$i+j \quad i-j \quad i\times j \quad i\div j$$

の計算をして結果を表示するプログラムは次のように書きます.

```
// example 2.3
#include <iostream.h>

main( )
{
    int i,j;
    cout << "i=";
    cin >> i;           // i を読み込む
    cout << "j=";
    cin >> j;           // j を読み込む
    cout << "i+j=" << i+j << "¥n";
    cout << "i-j=" << i-j << "¥n";
    cout << "i*j=" << i*j << "¥n";
    cout << "i/j=" << i/j << "¥n";
}
```

乗算記号

乗算は記号 * で表します. 乗算記号の省略はできません. たとえば,

(正) 2*a　(正) (a+b)*(c+d)

(誤) 2a　(誤) (a+b)(c+d)

除算記号

除算は記号 / で表します (÷ という記号は使えません).

【実行例】

```
i=10
j=2
i+j=12
i-j=8
i*j=20
i/j=5
```

cout の右に式を書ける

式の計算の結果を変数に代入しないで，直接出力することができます．

（例） `cout << a+b;`

【注意】 整数型の数の割り算は，**整除**，すなわち

答（商）が整数で表される所まで割る

という規則で計算されます．

変数名の付けかた

◎ 変数名に使用できる文字は

小文字 `abcdefghijklmnopqrstuvwxyz`

大文字 `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

数字 `0123456789`

下線 `_`

◎ 小文字と大文字は「別の字」として扱われます．

慣習としては，普通は小文字だけを使い，大文字は特別な場合だけに使います．

◎ 先頭に数字を使ってはいけません．

◎ 使えない綴り（予約語）が少しあります（68ページ参照）．

◎ 字数制限は「事実上ない」と思ってよいでしょう*.

* TurboC++ では長さ制限ナシ．ただし識別には先頭の32字まで有効．
Microsoft C++ では247文字以内．

プログラム例 2.4——マクドナルド

単価 a 円のハンバーガを n 個買って y 円札を出したら，お釣りはいくらになるでしょうか？

この計算をするプログラムは次のように書きます．

```
// example 2.4
#include <iostream.h>>

main( )
{
    int a,n,y;
    cout << "a=";    cin >> a;
    cout << "n=";    cin >> n;
    cout << "y=";    cin >> y;
    int x=y-a*n;
    cout << "お釣りは" << x << "円です";
}
```

【解説】

◆ この種の問題は整数で計算できるので，変数 a, n, y を整数型と宣言しました（int a,n,y;）.

◆ 実行の際，各時点でキーボードから何を入れたらよいか迷わないように

```
cout << "a=";    cout << "n=";
cout << "y=";
```

によってガイド・メッセージを表示しています．

◆ 変数 x の型宣言は計算の直前に書いてあります．C++では，このように必要になった時点で型宣言を行なうことができます．

◆ 最後の cout では，まず「お釣りは」が表示され，続いて計算結果 x の値が表示され，最後に「円です」が表示されます．

---

* C ではこのような書き方ができなかったので，長いプログラムを書くときには，変数を使うたびにメモしておいて，冒頭の宣言文にまとめて書く必要がありました．

【実行例】

```
a=390
n=2
y=1000
お釣りは220円です
```

整数型の定数

整数型定数は，数学における普通の書き方と同様，数字を並べて表します．

(例) `365` `1993`

8進法および16進法の定数の書き方

> 0（ゼロ）で始まる「数字の列」は8進法の定数と見なされます．

(例) `0473` は $4 \times 8^3 + 7 \times 8^2 + 3 \times 8^1$ を表します．

> 頭に 0x（ゼロエックス，エックスは大文字でも小文字でもよい）を付けた「16進数字の列」は16進法の定数と見なされます．

16進数では

A が10　B が11　C が12　D が13　E が14　F が15

を表します．

(例) `0X12A` は $1 \times 16^3 + 2 \times 16^2 + 10 \times 16^1$ を表します．

余りの表し方

m を n で割った余りは　m%n　で表します.

（例）　c=d%7;　　c に「d を 7 で割った余り」を代入する.

長い整数と短い整数

整数を扱う型には，詳しくいうと

short int	短い整数	普通は語長16ビット
long int	長い整数	普通は語長32ビット
int	整数	普通は語長32ビット

の 3 種類があり，それぞれに

signed	符号付き
unsigned	符号なし

があります（黙っていれば signed になりますので，「符号なし」にしたい場合だけ，

```
unsigned int i,j,k;
```

というように，頭に unsigned を付けます）.

普通は int を使いますが，システムによっては int の語長が短いもの（16ビット）があるので，長い整数を扱うために long という型が設けられました. unsigned は，論理演算，画像データ，コード（符号）などによく用いられます.

インクリメント演算子

カウント（回数を数える）などの目的で，

整数型の変数 **i** の内容に 1 を加えたい

ということがよくあります．他の言語だと，これを

i=i+1

で表しますが，C や C++ では簡単に

```
++i;
```

と書くことができます．このようにすると，変数名 **i** が 1 回しか現れないのでコンパイラがその解析をする手間が省けますし，プログラムを書く人にとっても（特に変数名が長い綴りの場合）少し楽になります．

同様に「変数 **i** の内容から 1 を引きたい」という場合は

```
--i;
```

と書きます．なお，これに似た記法で

i++　　**i--**

というのがあります．意味は **++i** や **--i** とほとんど同じですが，途中代入のような形で使われた場合，

i++　**i--**　ならば　「1 を加減する前の値」

++i　**--i**　ならば　「1 を加減した後の値」

が式（など）の計算に使われます．

2.3 実数の計算

プログラム例2.5 —— $a+b=c$

二つの実数 a, b を読み込み,

$$a+b=c$$

の計算をして表示するプログラムは次のように書きます.

```
// example 2.5
#include <iostream.h>

main( )
{
   float a,b;
   cout << "a=";
   cin >> a;          // aを読み込む
   cout << "b=";
   cin >> b;          // bを読み込む
   float c=a+b;            // 計算
   cout << "c=" << c << "¥n"; // cを表示
}
```

【実行例】

```
a=123
b=567
c=690
```

実数型変数の宣言

メイン・プログラムの本文の冒頭にある

```
float a,b;
```

は「変数 a, b は実数型である」という宣言です.

定数の書き方

実数型の定数は，小数点を付けて書きます（小数点を付けないと整数型定数になってしまいますので注意して下さい）．

（例）　`3.14`　　`120000.0`
　　　`-3.14`　　`-120000.0`

非常に大きな数や非常に小さな数を表すときには，普通の定数の後に

　　`e` 符号 指数

を付けて「$\times 10^n$」を表す方式（指数付き表現）を用いると便利です．

（例）　`123.45e6`　　(123.45×10^6 を表す)
　　　`123.45e-6`　(123.45×10^{-6} を表す)
　　　`78e9`　　　　(78×10^9 を表す)

整数型の変数との間の型変換

整数型と実数型を混ぜて使うと，整数型の値を実数型に変換して計算されます．違う型の変数に代入するときも自動的に変換されます．

（例）　`a=i;`　　　整数型変数 `i` の値を実数型変数 `a` に代入
　　　`c=j+b;`　　整数型変数 `j` と実数型変数 `b` を加算する

キャスト

型変換を明示的に書きたい場合は，左に（型名）を付けて表します．

　　`float(i)`　　整数型変数 `i` を実数型に変換して使う場合
　　`int(a)`　　　実数型変数 `a` を整数型に変換して使う場合

この書き方をキャストといいます．これを次のように表すこともあります*.

```
(float)i
(int)a
```

* C ではいつもこのように書いていました．C++ では型名の後に ( ) を付ける方式を推奨しています．

単精度と倍精度

詳しくいうと，floatは「単精度（有効桁数約7桁）の浮動少数点表現の変数」であることを表しています．実数を扱う型としては，そのほかに

倍精度型　　double

拡張倍精度型*　long double

があります．

CやC++では「実数演算は原則として倍精度で行なう」という規則になっているので，floatで計算してもdoubleで計算しても，計算時間はあまり変わりません．そのため，特に単精度で計算したい場合以外はdoubleを使うのが普通です．

用語解説

浮動小数点演算（実数演算）　数値を

指数部（exponent）　位取り情報を表す

仮数部（mantissa）　有効数字を表す

の組（ペア）として表現し，処理する演算方式で，小数点を含む数や非常に大きな数（たとえば10^{20}）を扱うことができます．

（例）　$10^1 \times 0.31416$

指数部	仮　数　部

* 拡張倍精度型をサポートしていない処理系もあります．その場合，longを付けてもエラーにはなりませんが，精度はdoubleと同じです．

仮数部の桁数

機種によって多少は差がありますが，普通は

単精度	のとき24ビット（10進法に換算して約７桁）
倍精度	のとき52ビット（10進法に換算して約16桁）
拡張倍精度	のとき64ビット（10進法に換算して約20桁）

と考えてよいでしょう．

表現できる大きさの範囲

これも機種によって差がありますが，普通は

単精度	のとき10^{38}ぐらいまで
倍精度	のとき10^{308}ぐらいまで
拡張倍精度	のとき10^{4932}ぐらいまで

と考えてよいでしょう（IEEE 方式の場合）．

定数の書き方（続）

普通に書いた実数型の定数は（特に語尾に精度指定文字を書かなければ）自動的に倍精度（double）の扱いになります．

> 単精度の定数にしたい場合は，末尾に f または F を付けます．
> 拡張倍精度の定数にしたい場合は，末尾に l または L を付けます．

（例）

```
3.1416F    1.0e8F    単精度
3.1416     1.0e8     倍精度
3.1416L    1.0e8L    拡張倍精度
```

プログラム例2.6——倍精度計算

二つの実数 a, b を読み込み，倍精度で

$$a+b \quad a-b \quad a\times b \quad a\div b$$

の計算をして結果を表示するプログラムは次のように書きます．

```
// example 2.6
#include <iostream.h>

main( )
{
    double a,b;
    cout << "a=";
    cin >> a;              // aを読み込む
    cout << "b=";
    cin >> b;              // bを読み込む
    double wa=a+b;                // 和
    double sa=a-b;                // 差
    double seki=a*b;              // 積
    double shou=a/b;              // 商
    cout << "和 " << wa << "¥n"; // 表示
    cout << "差 " << sa << "¥n"; // 表示
    cout << "積 " << seki << "¥n"; // 表示
    cout << "商 " << shou << "¥n"; // 表示
}
```

【解説】 型宣言が double になっている点を除けば，普通のプログラムと全く変りませんね．これが C++ のうれしい所です（C の場合には，入出力の際に型の指定が必要になります）．

演算記号（+−*/）は整数型の場合と同じです（ただし，余りを表す演算子「%」は整数型でないと使えません）．

【実行例】

```
a=30
b=24
和54
差6
積720
商1.25
```

プログラム例2.7——拡張倍精度計算

実数 a を読み込み，拡張倍精度で

$$a+\pi \quad a-\pi \quad a\times\pi \quad a\div\pi$$

（ただし π は円周率）の計算をして結果を表示するプログラムは次のように書きます．

```
// example 2.7
#include <iostream.h>

main( )
{
   long double a,wa,sa,seki,shou;
   const long double pi=3.14159265359L;
   cout << "a=";
   cin >> a;             // aを読み込む
   wa=a+pi;              // 和
   sa=a-pi;              // 差
   seki=a*pi;            // 積
   shou=a/pi;            // 商
   cout << "和 " << wa << "¥n";    // 表示
   cout << "差 " << sa << "¥n";    // 表示
   cout << "積 " << seki << "¥n";  // 表示
   cout << "商 " << shou << "¥n";  // 表示
}
```

【実行例】

```
a=10
和13.1416
差6.85841
積31.4159
商3.1831
```

定数を表す const

変数の型宣言のとき，前（左）に const と書くと「その変数の値は変わらない（定数である）」と解釈され，その変数には代入ができなくなります．上の例では円周率 π をそのような形で扱っています*．

---

* C では，そのような場合に

```
#define PI 3.1416
```

という書き方を用いていましたが，C++ では const で書くのが普通です．

多重代入

たとえば，$a+b$ という計算をして，その結果を c にも d にも入れたい場合，CやC++では次のように書くことができます．

```
d=c=a+b;
```

このようにすると，

```
c=a+b;      d=c;
```

とするよりも見易く（d にも a+b が入ることがよくわかる）時間的にも有利です（レジスタとメモリの間のデータ転送の時間を節約できるため）．

途中代入

式の計算の途中で「この中間結果はあとで別の計算にも使うから残しておきたい」と思った場合，CやC++では「途中で手を休めて代入」ということができます．たとえば，

```
x=(a+b)*(a-b);
y=(a+b)/(a-b);
```

と書くと a+b と a-b が2回ずつ計算されることになりますが，

```
x=(c=a+b)*(d=a-b);
y=c/d;
```

と書くことによってムダを省くことができます．途中代入は「文」ではないので，末尾に「;」を付けず，( ) で囲んで範囲を指示します．

インクリメント

47ページで説明したインクリメント演算子 ++ および -- は実数型の変数にも適用でき，それぞれ「1.0 を加え込む」および「1.0 だけ差し引く」の意味になります．

もとの場所への代入

インクリメントは変数に1を加える（または引く）場合でないと使えませんが，数値計算においては，「式の計算をして，その結果をある変数に加える」ということがよくあります．その場合，普通は

変数名 = 同じ変数名 + 式

の形で書きますが，CやC++ではこれを

変数名 += 式 ;

（例） `a+=x*y;`

と書くことができます．同様に

`-=`　`*=`　`/=`

などの記法を用いることができます．

まとめ

a に1を加えて a に代入	`++a`
a から1を引いて a に代入	`--a`
a の値に式の値を加えて a に代入	`a+=` 式
a の値から式の値を引いて a に代入	`a-=` 式
a の値に式の値を掛けて a に代入	`a*=` 式
a の値を式の値で割って a に代入	`a/=` 式

2.4　使用できる数学的関数

C++においては，CやFORTRANやBASICと同様に，

三角関数	sin, cos, tan
逆三角関数	asin, acos, atan
双曲線関数	sinh, cosh, tanh
指数関数	exp, pow
対数関数	log
平方根	sqrt

などの数学的関数を使用することができます．また，処理系によっては*

ガンマ関数　　ベッセル関数　　誤差関数

などの特殊関数も使用できます．どのような関数を使用できるかは，math.h というヘッダーファイルを表示してみると簡単にわかります．

一般的な注意

これらの関数を使う際には，プログラムの冒頭に

```
#include <math.h>
```

を書いておく必要があります**．

また，処理系によっては***，コンパイルする際に，

```
コンパイル指令　ファイル名　-ライブラリ名
```

(例)　cl abc.cpp -lm

という形式で指令することが必要になります．これはライブラリから数学関数のプログラムを取り出してきてリンク（結合）するためです．

---

* たとえば，Microsoft C++

** これを忘れると「…is not defined.」(この関数は定義されていない，すなわち型宣言を忘れている）というエラーメッセージが出るでしょう．

*** Turbo C の場合は不要．

関数の型と引数の型

Cの場合には，大部分の数学的関数が

引数の型は　　　　`double`（倍精度実数型）

関数の型（結果）も　`double`（倍精度実数型）

という形式で書かれていました．

しかし，C++では引数の型に合わせて関数が自動的に選択されて呼び出されます．たとえば，引数に拡張倍精度型の変数を書けば，拡張倍精度型の関数が呼出され，それなりに精度の高い計算がなされます．

角度の単位はラジアン

三角関数，逆三角関数の計算における角度の単位はラジアンです．度をラジアンに変換するには

$\pi/180 \fallingdotseq 0.017453292$

を掛けます．ラジアンを度に変換するには

$180/\pi \fallingdotseq 57.29577951$

を掛けます．π の精密な値は `math.h` の中で定数として定義してあるのが普通ですから，それを利用するとよいでしょう（たとえば，Turbo C++の場合には `M_PI` という名前になっています）．

三角関数，逆三角関数

```
sin(x)    asin(x)
cos(x)    scos(x)
tan(x)    atan(x)
          atan2(y,x)
```

◆ `atan2(y,x)`は，原点から見た座標 (x, y) の点の向きを計算する関数，いいかえれば

$\arctan(y/x)$

を計算する関数で，直角座標から極座標への変換などに使われます．結果の範囲は

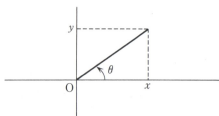
$-\pi \sim \pi$

になるのが普通です．

$\tan\theta = \frac{y}{x}$　　$\theta = \arctan\frac{y}{x}$

atan2 の引数の意味

双曲線関数，逆双曲線関数

```
sinh(x)    asinh(x)
cosh(x)    acosh(x)
tanh(x)    atanh(x)
```

指数関数，対数関数，平方根

標準的な関数

平方根	`sqrt(x)`
e^x を計算する	`exp(x)`
自然対数 $\log_2 x$ を計算する	`log(x)`
常用対数 $\log_{10} x$ を計算する	`log10(x)`
x^y を計算する	`pow(x,y)`

◆ `pow(x,y)` の `y`（べき指数）は `double` です．したがって $x^{1.3}$ というような値も計算できます．

絶対値など

整数の絶対値は	`abs(i)`
実数*の絶対値は	`fabs(x)`
複素数*の絶対値は	`cabs(z)`
床　（x 以下の最大の整数）	`floor(x)`
天井（x 以上の最小の整数）	`ceil(x)`
x に最も近い整数	`rint(x)`

* ただし，`i` の型は `int`，`x` の型は `double`，`z` は複素数を扱う構造体で，`z.x` が実数部，`z.y` が虚数部．

2.5　補　足

C との互換性

C では注釈を

> 複合記号 /* と */ に挟まれた部分は注釈とみなされる．

という形で扱ってきました．この書き方は C++ でも有効です．したがって，これまでの C で書かれたプログラムを流用する際に，注釈まで書き直す必要はありません．

C における入出力の関数

```
scanf( 書式 , 格納先を列挙 );
printf( 書式 , 出力項目を列挙 );
```

は C++ でも使えます．ですから，C で書かれたプログラムを書き直す必要はありません．

実数計算における「余り」の書き方

m を n で割った余りは m % n で表すことができますが，これは整数型専用です．x, y が実数型（float や double）のとき，x を y で割った余りを計算するには，関数

```
fmod(x,y)
```

を使います．

シフト

CやC++には「シフト演算子」があって，数値のビット列を左右に動かすことができます．コンピュータの内部では数が2進法で表されていますから，

左に n 桁シフトすれば 2^n 倍

右に n 桁シフトすれば $1/2^n$ 倍

したことになります．

mをn桁右シフト	m >> n
mをn桁左シフト	m << n

ビットごとの演算

ビットごとのAND	&
ビットごとのOR	|
ビットごとのXOR	^
ビット反転	~

【解説】「&」演算子は，マスクを掛けてビット列の一部分だけを取り出す操作によく用いられ，「|」演算子はその逆の操作（合成）などに用いられます．XOR（exclusive-OR，排他的論理和）は

「両方が0」または「両方が1」ならば結果は0

「一方が1」で「もう一方が0」ならば結果は1

という処理で，「^」演算子はこれをビットごとに行ないます．

オーバーフロー（あふれ）

計算結果が数値の「表現できる範囲」を越えると「オーバーフロー」という状態になります．具体的には

　　整数演算の結果が表現できる桁数を越えた

　　実数演算の結果の指数部が表現できる桁数を越えた

などの場合に起こります．また，

　　0 で割った

場合も（これを zero divide といって区別する処理系もありますが）同様な状態になります．

これらの内，整数演算のオーバーフローは，たいてい，

　　はみだした部分（上位の桁）を無視して計算を続ける

ということになります．一方，実数演算の指数部オーバーフローは

　　エラーメッセージが出て停止する

のが普通です．

アンダーフロー

逆に計算結果が小さくなりすぎて「実数型で表現できる数の小さい方の限界」を越えると「アンダーフロー」というエラーになります．アンダーフローの扱いは処理系によって異なります．普通は

　　結果を 0 とみなして先に進む

ようですが，

　　エラーメッセージが出て停止する

ような機種もあります．

初期値の指定

CやC++においては，型宣言の際に「その変数の初期値」を指定することができます．これは

```
型名　変数名 = 初期値 ;
```

（例） `double s=0.0;`　　変数 s の初期値を 0 にする

または

```
型名　変数名 ( 初期値 );
```

（例） `int k(10);`　　変数 k の初期値を10にする

またはそれらを列挙した形，たとえば

```
int i=2,j=3;
int m(2),n(3);
```

のように書きます．初期値を指定しておけば，プログラムの実行を開始するとき（また，関数が呼び出されるときも）変数にその値が代入されます．

定数を表す修飾語 const

既に53ページでも説明したように，型宣言の前（左）に const と書いておけば，その変数の値は書き変えることができなくなります．定数はなるべくこのようにして扱うのが安全です．

（例） `const double e=2.718281828;`

列挙型

CやC++には整数型や実数型のほかに列挙型という型があります．これは，たとえば

　　月，火，水，木，金，土，日
　　クラブ，ダイヤ，ハート，スペード

などのように，

　　数でないものを扱う
　　扱う対象が限定されている
　　したがって，全部を列挙できる

という場合に使われる型です．これを使うには，まず

enum　型の名前　{ 値を全部コンマで区切って列挙 };

(例)　`enum janken {guu,choki,paa};`

の形式で「型の名前」を宣言します．型の名前というのは，たとえば

　　曜日を扱うために　　　　`week`
　　トランプを扱うために　　`card`

というように適当に名前を付けるわけです．そして，変数の型宣言は

型の名前　その型で扱う変数名をコンマで区切って列挙 ;

(例)　`janken a,b;`

の形で書きます*.

* Cでは型の名前の前に「enum」を書く規則になっていましたがC++ではその必要がありません．

こうして宣言された変数は，その名前の列挙型（いうなれば，ジャンケン型，曜日型）の変数として，代入したり，比較したり（これについては次章で説明します），関数に引き渡したりすることができます．enum の {} 内に列挙した値は「定数」として使用できます．

列挙型の入出力は，あまり円滑にいきません．cout を使えば，一応出力してくれますが，値は整数値に化けてしまいます．その値は列挙型宣言のときに {} 内の最初に書いた値が 0，その次に書いた値が 1，次が 2，…という対応になるので解読することは可能ですが，見やすくするには，たとえば

```
enum janken {guu,choki,paa} a;
char gcp[3][4]={"ｸﾞｰ","ﾁｮｷ","ﾊﾟｰ"};
a=choki;
cout << gcp[int(a)] << '¥n';
```

というような処理が必要になります．

数値型，列挙型以外の型

整数型，実数型，列挙のほかに，次のような型があります．

char	文字型	語長 8 ビット
void	空型	実体ナシ

文字型の変数には文字（正確にいえば英字，数字など，いわゆる半角文字）を 1 字だけ記憶させることができます．もっと長い文字列を扱いたい場合には「文字型配列」の形で扱います．文字や文字列の扱い方については，後の 4 章で詳しく説明します．

空型（void）は特別な型で，5 章で説明します．

複素数の計算

CやC++には「基本的な型」としての複素数型はありません．しかし，C++には「クラス」という書き方があって，変数の型を自分で定義して使用できるようになっています．大部分の処理系では，その機能を応用して，

complex

という「複素数の計算をするためのクラス」を提供しています．これを使えばあたかも複素数型があるかのようにプログラムを書くことができます．

それには，まず，プログラムの最初に

```
#include <complex.h>
```

と書きます*．複素数を代入する変数は，

```
complex 変数名 ;
```

(例)　`complex a,b,c;`

の形で宣言します．複素数の**定数**は

```
complex( 実数部 , 虚数部 )
```

(例)　`complex(1.2,3.4)`　　$1.2+3.4i$ のこと

で表します．

* これは文法ではなくて，大部分のクラス・ライブラリの仕様がこのようになっている，という説明です．

演算記号 **+-*/** や代入記号 **=** や **( )** などは整数型や実数型と同様に使用できます．入出力も，整数型や実数型と同様に，

```
cin >> 変数名;
cout << 変数名;
```

と書けます．キーボードから複素数の値を入れるときには，

(実数部 **,** 虚数部 **)** ⏎

(例)　2 + 3*i* を入れる場合　**(2,3)** ⏎

の形で打鍵します．

プログラム例2.8――複素数の計算

複素数で **c3=c1+c2** の計算をして結果を表示するプログラムは次のように書きます．

```
//   example 2.8

#include <complex.h>
#include <iostream.h>

main( )
{
  complex c1,c2,c3;

  cin >> c1;              // 複素数を読み込む
  c2=complex(2.0,3.0);           // 2+3i を c2 に代入
  c3=c1+c2;               // 複素数の加算
  cout << c1 << '¥n';     // c1 を出力
  cout << c2 << '¥n';     // c2 を出力
  cout << c3 << '¥n';     // c3 を出力
}
```

予約語

下記の綴りは文法上特別な意味をもつので，名前（変数名など）には使用できません．

asm	float	return
auto	for	short
break	friend	signed
case	goto	sizeof
catch	huge	static
cdecl	if	struct
char	inline	switch
class	int	template
const	interrupt	this
continue	long	throw
default	near	try
delete	new	typedef
do	operator	union
double	pascal	unsigned
else	private	virtual
enum	protected	void
extern	public	volatile

以上のほかに，下線（_印）で始まる予約語がたくさんあります．下線で始まる綴りは「特別な名前」としてシステム用によく使われますので，使用を避けておく方が安全です．

3

くりかえしや場合分け処理の書き方

この章では，

大小の判定

場合分け処理

一定回数のくりかえし

条件が満たされるまでのくりかえし

添字付き変数

自作の関数

などの書き方を説明します．

制御文に関してはCとほとんど同じなので，既にCをよくご存知の方は，ざっと目を通すだけで十分です．

3.1 if 文

プログラム例3.1――ディスカウント

A社のカセット・テープC60は1本300円ですが，10本を超える分については1本270円で売ります．本数 *n* に対する金額を計算するプログラムを作ってみましょう．

```
// example 3.1
#include <iostream.h>

main( )
{
  int n,yen;
  cout << " n= ";
  cin >> n;
  if ( n<=10 ) yen=300*n;
  else         yen=3000+270*(n-10);
  cout << " 金 額 は  " << yen << '¥n';
}
```

【実行例】

```
n= 7
金額は 2100

n= 12
金額は 3540
```

C の if 文は次の形式で書きます．

```
if ( 条件式 ) 条件成立の場合に実行すべき文 ;
     else     それ以外の場合に実行すべき文 ;
```

実行すべき文が二つ以上ある場合は，それらを {} で囲んで書きます．

(例)
```
if (x>0.0) {y=x; z=0.0;}
     else  {y=0.0; z=1.0;}
```

条件式としては，等式や不等式を書くことができます．等号や不等号は次のように表します．

<	より小さい	<=	より小さいか等しい（以下）
>	より大きい	>=	より大きいか等しい（以上）
==	等しい		
!=	等しくない		

この内，特に注意を要するのは等号です．

```
✕ if (a=b) …
  if (n=3) …
```

つい，うっかり，このように書いてしまうのですが，このように書くとCの文法では

「a に b を代入し，その値が 0 でなければ…」

「n に 3 を代入し，その値が 0 でなければ…」

と解釈されてしまいます．

【他の言語に慣れている人のための注意】 PASCAL では `else` の前に `;` を付けるとエラーになりますが，C や C++ では逆に `;` を付けないとエラーになります．

プログラム例3.2——2次方程式の根

2次方程式

$$ax^2 + bx + c = 0 \quad (\text{ただし } a \neq 0)$$

の根を求めるプログラムを作ってみましょう.

```
// example 3.2
#include <iostream.h>
#include <math.h>
main( )
{
  double a,b,c,d,bunbo;
  double r,s;
  double x,x1,x2;
  cout << "a=";  cin >> a;
  cout << "b=";  cin >> b;
  cout << "c=";  cin >> c;
  d=b*b-4.0*a*c;
  bunbo=2.0*a;
  if (d>0) {
    x1=(-b+sqrt(d))/bunbo;
    x2=(-b-sqrt(d))/bunbo;
    cout << "x1=" << x1 << '¥n';
    cout << "x2=" << x2 << '¥n';
  } else if (d==0.0) {
    x=(-b)/bunbo;
    cout << "x=" << x << '¥n';
  }
  else {
    r=(-b)/bunbo;
    s=sqrt(-d)/bunbo;
    cout <<"x1="<<r<<'+'<<s<<"i ¥n";
    cout <<"x2="<<r<<'-'<<s<<"i ¥n";
  }
}
```

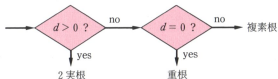
【解説】

1) 冒頭の `#include <math.h>` は sin, cos, log などの数学的標準関数を使うときの決まり文句で，今ここでは平方根を計算する関数 `sqrt` を使用するためです．

2) 2 次方程式の根の公式は

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

ですね．このプログラムでは，根号の中の $b^2 - 4ac$ が正の場合，0 の場合，負の場合に分けて処理しています．

3) `else` の中にまた `if` が入る 2 段階の分岐ですが，ここに示したように `else if` という形で書くことができます．

【実行例】

```
a=2
b=-10
c=12
x1=3
x2=2
```

```
a=3
b=-6
c=3
x=1
```

```
a=3
b=3
c=3
x1=-0.5+0.866025i
x2=-0.5-0.866025i
```

if 文の文法のマトメ

```
if ( 条件式 )  条件式が成立した場合に実行すべき文 ;
    else       条件式が成立しなかった場合に実行すべき文 ;
```

「実行すべきこと」が一つの文で書けない場合は次のように書きます.

```
if ( 条件式 ) {

        条件式が成立した場合に実行すべきプログラム

} else {

        条件式が成立しなかった場合に実行すべきプログラム

}
```

【注意】

◆ 実行すべき文の終りにはいつでも「;」を付けます.

◆ それらを囲む {} の後には「;」を付ける必要がありません.

◆ 他の言語と違って「then」や「endif」は不要です.

等号，不等号の書き方

等しい	**==**	**等しくない**	**!=**
大きい	>	以上	>=
小さい	<	以下	<=

【注意】

◆ 等号が「=」でなくて「==」であることに注意して下さい.

うっかり「=」を用いると「代入」されてしまいます！

論理演算子

(例)　`!((i+j)>2)`　　`i+j` が 2 より大きくない

　　　`(m==0)&&(n==0)`　m, n がともに 0

　　　`(a<=1)||(a>=3)`　1 以下または 3 以上

多重使用

```
if ( 条件式 ) {
     .......
     .......
     .......
}
else if {
     .......
     .......
     .......
}
else {
     .......
     .......
     .......
}
```

というような構文を用いることもできます（プログラム 3.2 参照）.

選択代入

if 文を使うかわりに

変数名 = (条件式) ? 式1 : 式2 ;

という書き方を用いることもできます．

条件式成立の場合は式 1

条件式不成立の場合は式 2

が実行されます．

(例)　`c=(a>b)?a:b;`

と書けば，

$a > b$　ならば　$c = a$

$a \leqq b$　ならば　$c = b$

となりますから，結局，a と b の内，大きい方が c に代入されます．

3.2 while 文

プログラム例3.3 — 1 + 2 + … + 10

1から10までの整数の和

$$S = 1 + 2 + \cdots + 10$$

を計算するプログラムを作ってみましょう．

```
// example 3.3
#include <iostream.h>

main( )
{
  int i=1,s=0;
  while(i<=10) {
    s+=i;
    ++i;
  }
  cout << "s=" << s << '¥n';
}
```

開始

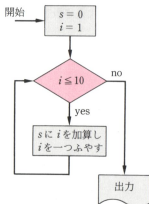
$s = 0$
$i = 1$

$i \leqq 10$

no

yes

sにiを加算し
iを一つふやす

出力

【解説】

1)　くりかえしには while 文を使います．書きかたは

```
while( 条件式 ) くりかえし実行すべき文 ;
```

または

```
while( 条件式 ) {
        くりかえし実行すべきプログラム
}
```

で，条件式が成立している限り {} 内がくりかえし実行され，条件式が不成立になれば反復を終了して次の文に進みます．

2)　複号記号 += と ++ は C や C++ 独特の書きかたで

+=　は「右辺の値を左辺の変数に加え込む」

++　は「その右の変数の値を一つふやす」

ことを表します．

3)　s は部分和を入れる場所として使っています．最初はそこに 0 を入れておきます．i は最初 1 ですが，反復のたびに 1 が加えられるので 1, 2, 3,… と10まで変っていきます．$i=10$になったとき，まだ$i \leqq 10$なので {} 内が実行され，その次の判定で no になって出力に進みます．

【実行例】

```
s=55
```

条件が成立している間だけ反復するための書き方は二通りあります．

1)　ループの冒頭で終了判定を行なう形：

```
while ( 条件式 ) {

    実行すべきプログラム

}
```

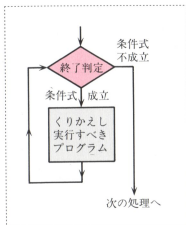
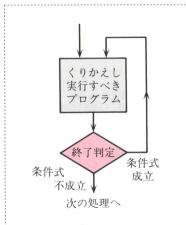
2)　ループの最後に終了判定を行なう形：

```
do {

    実行すべきプログラム

} while ( 条件式 );
```

概念図

while 文

do-while 文

while 文は，このようにループに入る前に終了判定を行ないますので，最初から「反復条件」が満たされてなければ，一回も「反復部分」を実行しないで終了となります．

【注意】

- 実行すべき文の終りにはいつでも「;」を付けます．
- それらを囲む {} の後には「;」をつける必要がありません．
- 条件式には，等式，不等式，論理式などが書けます（75ページ参照）．
- ( ) 内に書くのは「反復を続けるための条件」です．初心者はよくここに「反復を打ち切るための条件（たとえば収束判定のための不等式）」を書いて失敗しますので，注意して下さい．

3.3　for　文

プログラム例3.4——1 ＋ 2 ＋ … ＋ 10の別解

例 3.3 と同じ問題（1 から10までの整数の和

$$S = 1 + 2 + \cdots + 10$$

を計算するプログラム）は，for 文を用いて次のように書くことができます．

```
// example 3.4
#include <iostream.h>

main( )
{
  int s=0;
  for (int i=1; i<=10; ++i) {
    s+=i;
  };
  cout << s << '¥n';
}
```

【解説】

1)　C の for 文は BASIC や PASCAL の for 文とかなり違います．書き方は

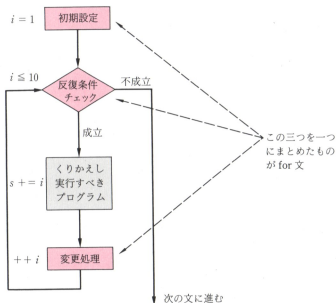
```
for ( 初期設定 ; 反復条件 ; 変更処理 )
        くりかえし実行すべき文 ;
```

または

```
for ( 初期設定 ; 反復条件 ; 変更処理 ) {
  くりかえし実行すべきプログラム
}
```

2) for 文を実行すると，まず「初期設定」の部分が実行され，次に「反復条件」がチェックされ，OK となれば「くりかえし実行すべきプログラム」が実行され，そのあとで「変更処理」がなされて，「反復条件のチェック」にもどります．

【実行例】

```
55
```

for 文のマトメ

```
for ( 初期設定 ; 反復条件 ; 変更式 ) {

    くりかえし実行すべきプログラム

}
```

（例 1）　i を 0, 1, 2, 3, 4, 5 の順に変えて実行

```
for ( i=0 ; i<=5 ; ++i )
    .......
```

（例 2）　x を 1 から64まで毎回 2 倍しながら反復

```
for ( x=1.0 ; x<=64.0 ; x*=2.0 )
    .......
```

【解説】

◆　「初期設定」というのはループに入る直前に実行する準備作業のことで，普通はここで「制御変数」に初期値を代入します．

◆　「反復条件」というのは反復を続行する条件のことです．この条件は毎回の反復の冒頭でチェックされ（while 文と同じ），この条件が満たされなければ反復を打ち切ります．

◆ 「変更式」というのは，次の回に移る前に制御変数の値などを更新する式で，普通はここで計数（カウント）のため，制御変数に増分を加算します．

◆ 「初期設定」や「変更式」に二つ以上の文を書きたい場合はコンマで区切って列挙します．

（例 3） **i** は 5, 4, 3, 2, 1 の順に，
j は 1, 2, 3, 4, 5 の順に変えて実行したい場合

```
for ( i=5,j=1 ; i>0 ; --i,++j )
        .......
```

【注意】

初心者が for 文を使うと，よく無限ループに入って止まらなくなってしまうことがあります．その原因は，たいてい次の内のどれかです．

① 初期設定，終了判定，値更新の変数名が一致していない．

（例） `for(i=0;n<10;++k)`

このような誤りは，変数名をとりかえたときによく起こります．また，長い綴りの変数名の入力ミスということもあります．

② 不等号の向きが逆になっている．

（例） `for(i=10;i<0;--i)`

終了判定を「～になるまで」のつもりで書いてしまうと，このような失敗を招きます．

③ ループの中で制御変数の値を書き換えている．

(③の例)
```
for(i=0;i>10;++i) {
        ........
          i=2;
        ........
}
```

他人が作った（または自分が昔作った）複雑なプログラムを何回も手直ししていると，このようなミスを犯しやすいので注意しましょう．Cコンパイラはこの種の誤りに対して警告を出してくれません．

ループからの脱出

> for, while, do などのループの途中で反復を打切って反復部分の外に出るには，
>
> `break;`
>
> という文を用います．break 文は，たとえば
>
> `if ( 収束判定不等式 ) break;`
>
> というような形で用いられます．

細かい注意

for 文を使うとき，記号 {} と ; の付け方が初心者にとって最初ちょっとわかりにくいので，その要点をまとめておきましょう．

1) 「くりかえし実行すべきプログラム」が一つの文だけであれば，

```
for ( 初期設定 ; 反復条件 ; 変更式 )
    くりかえし実行すべき文 ;
```

の形で書くことができます．この場合は {} は不要です．文の終りには，当然，セミコロン ; を付けます（これは「くりかえし実行すべき文」の終りの記号と for 文の終端記号を兼ねています）．入れ子になっても，この規則は通用します．たとえば，

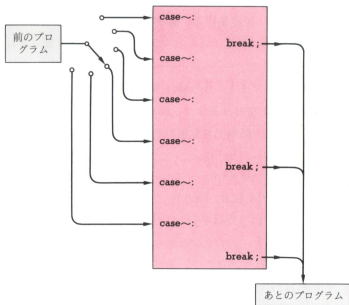
```
for(i=0;i<m;++i)
    for(j=0;j<n;++j)
        a[i][j]=i+j;
```

の内側のループを {} で囲む必要はありません．

2) 「くりかえし実行すべきプログラム」が二つ以上の文から成る場合には必ず {} で囲まなければいけません．個々の文の終わりには必ず ; 印を付けます．{} 内の最後の文は，次に } 印が来るのですから，いちいち ; を付けなくてもコンパイラが気をきかせればわかりそうなものですが，付ける規則になっています．

一方，{} の後には ; 印を付ける必要がありません（for 文の終端記号が必要な筈ですが，付けないでよいという規則になっています）．; 印を付けても誤りではありませんが，文法上は } 印のところで for 文が終わっており，そのあとの ; 印は「空文」とみなされます．

3.4　switch 文

これは一度にたくさんの道に分岐する，文字どおりのスイッチです．

書き方は次のとおりです．

switch (変数名) {

case 値$_1$:　その場合に実行すべきプログラム

break;

(以下同様にいくつでも列記)

default:　その他の場合に実行すべきプログラム

break;

}

プログラム例3.5――メニュー選択

円の面積，円周，球の体積の計算の内の一つを選んで実行させるプログラムを switch 文を使って書くと次のようになります．

```
// example 3.5
#include <iostream.h>

main( )
{
  const double pi=3.141593; /* 円周率  */
  const double c=4.0/3.0; /* 定数 4/3 */
  int m;             /* メニューの番号 */
  double r;          /* 半径       */
  cout << "次の計算ができます¥n";
  cout << "  1...円の面積¥n";
  cout << "  2...円周の長さ¥n";
  cout << "  3...球の体積¥n";
  cout << "どれにしますか ?";
  cin >> m;
  cout << "半径を入れて下さい¥n  r=";
  cin >> r;
  switch(m) {
    case 1 :   // 円の面積
      cout << "S=" << pi*r*r << '¥n';
      break;
    case 2 :   // 円周の長さ
      cout << "L=" << 2.0*pi*r << '¥n';
      break;
    case 3 :   // 球の体積
      cout << "V=" << c*pi*r*r*r << '¥n';
      break;
    default :
      cout << "計算できません¥n";
  }
}
```

【実行例】

```
次の計算ができます
  1...円の面積
  2...円周の長さ
  3...球の体積
どれにしますか ?2
半径を入れて下さい
 r=3
L=18.8496
```

```
次の計算ができます
  1...円の面積
  2...円周の長さ
  3...球の体積
どれにしますか ?1
半径を入れて下さい
 r=2
S=12.5664
```

【解説】 switch 文は一般に次の形が許されます．

```
switch( 式 ) {
  case 定数式1 :
        ここにいくつも文を書けます．
        普通は最後に break; を書きます．
  case 定数式2 :
        以下同様にいくつものケースの処理を列記します．
  default :
        上のどれにも該当しない場合に実行すべきプログラム
}
```

ここの「式」や「定数式」として書けるのは

　　整数型　　　文字型*　　　列挙型

だけです．

break は「ここで switch 文の実行を終了して先に進め」という意味です．もしもそれを書いておかなければ，次のケースの実行部分へ進みます．たとえば前のページのプログラム例の break 文を全部抜いてしまうと，$m=1$ の場合，円の面積，円周，球の体積の全部が出力され，最後の「アキマヘンデ」も出力される，ということになるでしょう．

---

* 7 章で説明します．

4

配列と文字データの扱い方

この章では，配列と文字データの使い方について説明します．配列というのは，要するに表（ひょう）のことで，一般には

数表，統計データ，帳簿

などを扱うために，また理工系では

数列，行列，ベクトル

などの計算のために必要になります．

一方，文字データは，たとえば

氏名，住所，品名

などを扱うのに必要です．

4.1 表の使い方

表のことをコンピュータ用語では「配列」といいます．正確にいえば，

　同じ型のデータを

　添字（番号）を付けて

　1列または四角（一般に多次元の直方体）に並べたもの

を配列といいます．

1次元配列

□ □ □ □ □ … □ □

2次元配列

□ □ □ □ □ … □ □
□ □ □ □ □ … □ □
□ □ □ □ □ … □ □
………………………………………
□ □ □ □ □ … □ □
□ □ □ □ □ … □ □

添字付き変数

配列の個々の要素（成分）を指示するには，配列名の後に [ ] で囲んで添字を付けて表します．C++の添字は0番から始まります．

数学の書き方：	a_0	a_1	a_2	…	a_n
C++の書き方：	a[0]	a[1]	a[2]	…	a[n]

注意を要するのは，二つ以上の添字を付ける場合で，

> 2次元配列の場合は，
> 　配列名 [第1添字] [第2添字]
> 3次元配列の場合は，
> 　配列名 [第1添字] [第2添字] [第3添字]

の形式で書きます（何次元でも以下同様）．BASICやFORTRANでは，添字をコンマで区切ってカッコの中に書きますが，C言語やC++では，添字を一つずつ [ ] で囲むのです．

（例）　3行3列の場合．

	第0列	第1列	第2列
第0行	a[0][0]	a[0][1]	a[0][2]
第1行	a[1][0]	a[1][1]	a[1][2]
第2行	a[2][0]	a[2][1]	a[2][2]

添字に式も書ける

[ ] の中には変数や式を書くことができます．

（例） `a[i]=b[i+1]+c[i*j-k];`

配列の宣言

配列を使用する場合は，プログラムの冒頭で配列の宣言をしておく必要があります．書き方は，

1 次元配列の場合：
　　型名　配列名 [要素の個数];
　　（例） `float a[10];`
2 次元配列の場合：
　　型名　配列名 [行数] [列数];
　　（例） `float a[10][10];`
3 次元配列の場合：
　　型名　配列名 [語数] [語数] [語数];
　　（例） `float a[10][10][10];`

（以下同様）です．同じ型の配列は，一つの行にコンマで区切ってまとめて宣言してもかまいません．

（例） `int b[29],c[60],d[51],wa[8][8];`

【注意】 添字は0番から始まりますので，使用できる添字の上限は宣言した「個数」の一つ手前までです．たとえば，

```
int b[100];   (または float b[100];)
```

と宣言した場合，使用できるのは

```
b[0], b[1], …, b[99]
```

の100個で，b[100] は使えません．b[100] まで使いたい場合は

```
int b[101];   (または float b[101];)
```

のように宣言する必要があります．

一般に，配列の宣言の [ ] の中には

使用したい添字の上限プラス1

を書くわけです．この規則はBASIC, FORTRAN, PASCAL などと全く違いますので，従来の言語に慣れている人は特に気をつけて下さい．

初期値の指定

配列宣言の中で初期値を指定することができます．書き方は*

```
型名 配列名 [ 寸法 ] = { 第0要素の初期値 , 第1要素の初期値 ,
                        第2要素の初期値 , 第3要素の初期値 ,
                        … , 最後の要素の初期値 } ;
```

(例) `float a[5]={3.6,0.8,7.9,2.8,-1.2};`

* 初期値として式を書くことも許されています．

4.2　1 次元配列

プログラム例4.1――平均値と標準偏差

n 個のデータ $a_1, a_2, \cdots, a_n$ の平均値と標準偏差を計算するプログラムは次のように書きます.

```
// example 4.1
#include <iostream.h>
#include <math.h>
#include <process.h>

main( )
{
   const int N=100;       // 配列の寸法
   const char CR='¥n';    // 改行記号
   int i,n ;
   double w ;             // 作業場所
   double a[N];           // データ
   // データの読み込み
   cout << "n=";   cin >> n;
   if (n>N) {
     cerr << " memory over ¥n";
     exit(1);
   }
   for (i=0; i<n; ++i) {
     cout << "a[" << i << "]=";
     cin >> a[i];
   }
   // 平均値の計算
   double s=0.0 ;
   for (i=0; i<n; ++i) {
     s+=a[i];
   }
   double heikin=s/n;
   cout << " 平均値 " << heikin << CR;
   // 分散の計算
   double ss=0.0;
   for (i=0; i<n; ++i) {
     w=a[i]-heikin;
     ss+=w*w;
   }
   double bunsan=ss/double(n);
   cout << " 分散 " << bunsan << CR;
   // 標準偏差の計算
   double sigma=sqrt(bunsan);
   cout << " 標準偏差 " << sigma << CR;
}
```

【解説】 このプログラムの計算式は

$$s = \sum_{i=1}^{n} a_i$$

$$(\text{平均値}) = s/n$$

$$ss = \sum_{i=1}^{n} (a_i - \text{平均値})^2$$

$$(\text{分散}) = ss/n$$

$$(\text{標準偏差}) = \sqrt{(\text{分散})}$$

です．記号 **+=** の意味は前に（55ページ）説明したように「右辺の値を左辺の変数に加え込む」ということです．

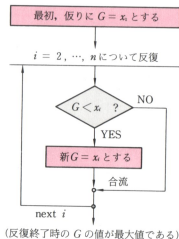
【実行列1】

```
n=3
a[0]=1.23
a[1]=4.56
a[2]=7.89
 平均値 4.56
 分散 7.3926
 標準偏差 2.71893
```

【実行例2】

```
n=5
a[0]=1
a[1]=2
a[2]=3
a[3]=4
a[4]=5
 平均値 3
 分散 2
 標準偏差 1.41421
```

◆　一組の値 $x_1, x_2, \cdots, x_n$ の最大値 G を求めるには次のようにします．

（反復終了時の G の値が最大値である）

◆　最小値も同様な要領で求めることができます．

プログラム例4.2―――最大値，最小値

n 個のデータ $a_1, a_2, \cdots, a_n$ の最大値と最小値を求めるプログラムを作ってみましょう．

```
// example 4.2
#include <iostream.h>
#include <math.h>
#include <process.h>

main( )
{
  const int N=100;
  int i,n ;
  double min,max;
  double a[N];
  // データの読み込み
  cout << "n=";    cin >> n;
  if (n>N) {
    cerr << " memory over ";
    exit(1);
  }
  for (i=0; i<n; ++i) {
    cout << "a[" << i << "]=";
    cin >> a[i];
  }
  // 最大値、最小値を見つける
  min=a[0];
  max=a[0];
  for (i=1; i<n; ++i) {
    if (min>a[i]) min=a[i];
    else if (max<a[i]) max=a[i];
  }
  // 結果の表示
  cout << "最小値  " << min << '¥n';
  cout << "最大値  " << max << '¥n';
}
```

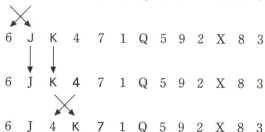
【実行例】

```
n=5
a[0]=753
a[1]=298
a[2]=815
a[3]=314
a[4]=606
最小値 298
最大値 815
```

勝手な順序で入力されたデータを大きさの順（番号順，成績順など）に並べかえることをソート（sort，整列）といいます．

それにはいろいろな方法がありますが，最も簡単なのは交換法です．

> **交換法*** 隣り合った二つのデータを比較し，
> 　　正しい順序ならばそのままにしておく
> 　　逆順になっていれば入れかえる
> という操作を，先頭から最後まで，何度もくりかえします．局部的な修正をするだけですから，一度や二度では完全にならないでしょうが最悪の場合でも $n-1$ 回（n はデータの個数）くりかえせば正しい順序になります．

お手もとにトランプがあったら，たとえばハートの札だけを13枚とり出し，よくきって札の上に並べ，上の手続きを実行してみましょう．

以下同様

* 水の中をアワが浮かび上がっていく感じに似ているので，「バブル・ソート」と呼ばれています．「本当のバブル・ソートはこれと違う」という説もありますが，大きな差異はなさそうです．

プログラム例4.3――バブル・ソート

n 個のデータ $a_1, a_2, \cdots, a_n$ を小さい順に並べかえるプログラムの最も簡単な1例を示します．

```
// example 4.3
#include <iostream.h>
#include <process.h>

main( )
{
  const int N=100;
  int       i,j,m,n;
  float     a[N],w;
  // データの読み込み
  cout << "n=";   cin >> n;
  if (n>=N) {
    cerr << " memory over ";
    exit(1);
  }
  for (i=0; i<n; ++i) {
    cout << "a[" << i << "]=";
    cin >> a[i];
  }
  // ソート
  for (m=n-1; m>=0; --m) {
    for (j=0; j<m; ++j) {
      if ( a[j]>a[j+1] ) {
        w=a[j];
        a[j]=a[j+1];
        a[j+1]=w;
      }
    }
  }
  // 結果の表示
  for (i=0; i<n; ++i) cout << a[i] << '¥n';
}
```

【実行例】

```
n=5
a[0]=256.1093
a[1]=947.2511
a[2]=262.5256
a[3]=293.3251
a[4]=252.8819
252.882
256.109
262.526
293.325
947.251
```

プログラム例4.4――フィボナッチ数列

出発値

$$a_0 = 1, \quad a_1 = 1$$

と漸化式

$$a_{n+1} = a_n + a_{n-1} \quad (n \geqq 2)$$

で生成される数列をフィボナッチ数列といいます．これを第10項（a_{10}）まで計算するプログラムを作ってみましょう．

```
// example 4.4
#include <iostream.h>

main( )
{
  const int N=11;
  int a[N];
  // 出発値の設定
  a[0]=1;        a[1]=1;
  // 漸化式
  for (int n=1; n<10; ++n)
    a[n+1]=a[n]+a[n-1];
  // 結果の表示
  for (n=0;n<=10;++n)
    cout << a[n] << '¥n';
}
```

【実行例】

```
1
1
2
3
5
8
13
21
34
55
89
```

【右のページのプログラムの実行例】

```
                        1
                      1   1
                    1   2   1
                  1   3   3   1
                1   4   6   4   1
              1   5  10  10   5   1
            1   6  15  20  15   6   1
          1   7  21  35  35  21   7   1
        1   8  28  56  70  56  28   8   1
      1   9  36  84 126 126  84  36   9   1
    1  10  45 120 210 252 210 120  45  10   1
```

プログラム例4.5——パスカルの三角形

パスカルの三角形というのは，2 項係数（いいかえれば，n 個の中から r 個をとる組合せの数）を漸化式によって計算するアルゴリズムで，最初に（いちばん上の段に）1 を書き，上の段から順に

隣合った二つの数を加えて下の段に書く

左端と右端に 1 を追加する

という規則で作ります．そのプログラムを書いてみましょう．

```
// example 4.5
#include <iostream.h>
#include <iomanip.h>

 main( )
{
  const int N=11;
  int a[N][N];
  // 出 発 値 の 設 定
  a[0][0]=1;  a[1][0]=1;  a[1][1]=1;
  // 漸 化 式
  for (int i=2; i<N; ++i) {
    for (int j=1; j<i; ++j)
      a[i][j]=a[i-1][j-1]+a[i-1][j];
    a[i][0]=1;  a[i][i]=1;
  }
  // 結 果 の 表 示
/*  cout << setfill(' ');   */
  for (i=0; i<N; ++i) {
    for (int k=0; k<2*N-2*i; ++k)
      cout << ' ';
    for (int j=0; j<=i; ++j)
      cout << setw(4) << a[i][j];
    cout << endl;
  }
}
```

プログラム例4.6 —— タテの合計ヨコの合計

A 中学の学年別，組別の数が，たとえば

	1 組	2 組	3 組	4 組	5 組
1 年	33	35	34	32	36
2 年	34	33	35	33	31
3 年	31	33	36	30	35

のように与えられたとき，各学年の人数および全校の人数を計算するプログラムは右のように書きます．

先に実行例を示しておきます．

```
1年1組33
1年2組35
1年3組34
1年4組32
1年5組36
2年1組34
2年2組33
2年3組35
2年4組33
2年5組31
3年1組31
3年2組32
3年3組36
3年4組30
3年5組35      ←ここまでが入力データ
1年は170人
2年は166人
3年は164人
全校では500人
```

【プログラム例】

```
// example 4.6
#include <iostream.h>

main( )
{
  const int N=5;  /* 組の数 */
  int a[3+1][N+1],b[3+1];
  /* データの読み込み */
  for (int i=1; i<=3; ++i) {
    for (int j=1; j<=N; ++j) {
      cout << i << "年" << j << "組";
      cin >> a[i][j];
    }
  }
  /* 計算 */
  int total=0;
  for (i=1; i<=3; ++i) {
    int subtotal=0;
    for (int j=1; j<=N; ++j) {
      subtotal+=a[i][j];
    }
    total+=b[i]=subtotal;
  }
  /* 出力 */
  for (i=1; i<=3; ++i) {
    cout << i << "年は" << b[i] << "人¥n";
  }
  cout << "全校では" << total << "人¥n";
}
```

プログラム例4.7——連立1次方程式

連立1次方程式を解くプログラムの作りかたを説明します．

ガウスの消去法 連立1次方程式の計算には普通，ガウスの消去法が用いられます．要領は筆算とだいたい同じで，それを表の形で規則的手順により計算を行います．

筆算の場合

（例）
$$\begin{cases} 2x + 4y + 6z = 6 \\ 3x + 8y + 7z = 15 \\ 5x + 7y + 21z = 24 \end{cases}$$

表にまとめる →

第1式を x について解くと

$$x = 3 - 2y - 3z$$

残りの式に代入して整理すると

$$\begin{cases} 2y - 2z = 6 \\ -3y + 6z = 9 \end{cases}$$

上の式を y について解くと

$$y = 3 + z$$

これを下の式に代入し整理すると

$$3z = 18 \quad \therefore \quad z = 6$$

順に上の式に代入して $y = 9, x = -33$ が得られます．

計算機による処理法

②	4	6	6
3	8	7	15
5	7	21	24

第1行を2で割り，3倍，5倍して第2行，第3行から引く．

1	2	3	3
0	②	−2	6
0	−3	6	9

第2行を2で割り，−3倍して第3行から引く．

1	2	3	3
0	1	−1	3
0	0	③	18

一般の場合　連立 1 次方程式を次の形に書きます.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

【計算手順】

前進消去
- $k = 1, 2, \cdots, n-1$ の順に
  - 新 $a_{kj} = a_{kj}/a_{kk}$　　$(j = k+1, \cdots, n)$
  - 新 $b_k = b_k/a_{kk}$
  - $i = k+1, \cdots, n$ について
    - 新 $a_{ij} = a_{ij} - a_{ik}a_{kj}$　　$(j = k+1, \cdots, n)$
    - 新 $b_i = b_i - a_{ik}b_k$

$x_n = b_n/a_{nn}$

後退代入
- $k = n-1, n-2, \cdots, 1$ の順に
  - $x_k = b_k - \sum_{j=k+1}^{n} a_{kj}x_j$

これを，そのとおりに C の文法に合わせて書けばよいわけですが，たとえば　新 $a_{kj} = a_{kj}/a_{kk}$ の計算は

```
a[k][j]/=a[k][k];
```

と書くことができ，これを $j = k, k+1, \cdots, n$ について計算するとき a[k][k] は共通ですから，あらかじめ p=a[k][k]; として

```
for(j=k;k<=n;++j)
    a[k][j]/=p;
```

とします. 他の式についても同様です.

【プログラム例】

```
// example 4.7
#include <iostream.h>
#include <process.h>

main( )
{
  const int NN=20;
  int i,j,k,n;
  double a[NN][NN];    /* 係数行列    */
  double b[NN];        /* 定数項      */
  double x[NN];        /* 解ベクトル  */
  double p,q,r,s;      /* 作業場所    */
  /* 入力 */
  cout << " nを入れて下さい ";
  cin >> n;
  if (n>=NN) {
    cerr << " memory over ";
    exit(1);
  }
  for (i=1; i<=n; ++i) {
    for (j=1; j<=n; ++j) {
      cout<<"a("<<i<<" ,"<<j<<")=";
      cin >> a[i][j];
    }
    cout<<"b("<<i<<")=";
    cin >> b[i];
  }
  /* 消去法の計算 */
  for (k=1; k<=n-1; ++k) {
    p=a[k][k];
    for (j=k+1; j<=n; ++j) {
      a[k][j]/=p;
    }
    b[k]/=p;
    for (i=k+1; i<=n; ++i) {
      q=a[i][k];
      for (j=k+1; j<=n; ++j) {
         a[i][j]-=q*a[k][j];
      }
      b[i]-=q*b[k];
    }
  }
```

```
    x[n]=b[n]/a[n][n];
    for (k=n-1; k>=1; --k) {
      s=0.0;
      for (j=k+1; j<=n; ++j) {
        s+=a[k][j]*x[j];
      }
      x[k]=b[k]-s;
    }
    /* 出力 */
    cout << "¥n 計算結果 ¥n";
    for (i=1; i<=n; ++i) {
      cout<<"x("<<i<<")="<<x[i]<<'¥n';
    }
    cout << "終り ¥n";
  }
```

【実行例】 例 4.7 の問題を解いてみます.

```
 nを入れて下さい 3
a(1 ,1)=2
a(1 ,2)=4
a(1 ,3)=6
b(1)=6
a(2 ,1)=3
a(2 ,2)=8
a(2 ,3)=7
b(2)=15
a(3 ,1)=5
a(3 ,2)=7
a(3 ,3)=21
b(3)=24

 計算結果
x(1)=-33
x(2)=9
x(3)=6
終り
```

4.3　文　字　型

文字データを扱う基本的な型は「文字型」です．文字型は一つの文字を表します．

宣言の書き方　　`char 変数名;`

(例)　`char moji;`

定数の書き方　　単引用符 ' で囲む

(例)　`'A'`

入出力の書き方

入力　`cin >> 文字型変数名;`

出力　`cout << 文字型変数名;`

(例)
```
cin  >> a;
cout << a;
```

条件式の書き方

◆　等号 == は「文字の一致」の意味で使えます．

(例)
```
if(moji=='A')
    printf("ヨクデキマシタ¥n");
```

◆　不等号 < や > を用いることもできます．その場合，大小は

A<B<C<……<Z

a<b<c<……<z

ｱ<ｲ<ｳ<……<ﾝ

0<1<2<……<9

となります（詳しくは113ページの表参照．そこに示されている番号の大小によって判定されます）．

プログラム例4.8——入出力と比較

文字を一つ読み込み，それがアルファベットの大文字であるか否かを判定するプログラムは次のように書けます．

```
// example 4.8
#include <iostream.h>

main( )
{
  char moji;
  cout<<"文字を一つ入れて下さい¥n";
  cin>>moji;
  if((moji>='A')&&(moji<='Z'))
    cout<<"アルファベットの大文字である¥n";
  else
    cout<<"アルファベットの大文字でない¥n";
}
```

【実行例】

```
文字を一つ入れて下さい
R
アルファベットの大文字である
```

```
文字を一つ入れて下さい
r
アルファベットの大文字でない
```

文字の表しかた

コンピュータの内部では，文字は普通0～255の番号（符号）で表されています．文字と番号の対応のさせかたは標準規格で決まっていて

　パソコン等では，たいてい，JIS (ASCII) コード

　大型計算機では，たいてい，EBCDIC (EBCDIK) コード

が使われています．参考のために，JIS コードの表の主部分を右のページに示します．0～31 は制御用コードで，

　　0　は「文字列の終り」の印（C, UNIX の場合）

　　11　は　tab（一定位置までのスキップ）

　　12　は　画面クリア，または「次のページに進む」

　　13　は　改行

を表します．漢字のコード体系はこれとは全く別で，英数字の2字ぶんの長さを使って表します．

【備考】

JIS = Japan Industrial Standard（日本工業規格）

ASCII = American Standard Code for Information Interchange（情報交換のための米国標準コード，アスキーと読む）

EBCDIC = Extended Binary Coded Decimal for Interchange Code（交換用拡張2進化10進符号，俗にエビスディックと発音される）

EBCDIK = Extended Binary Coded Decimal Including Kana（カナ文字を含む拡張2進化10進符号，これもエビスディックと読む）

ANSI = American National Standards Institute（米国規格協会，アンシーと読む）

JIS (ASCII) コード表

32		64	@	96	`	160		192	ﾀ
33	!	65	A	97	a	161	｡	193	ﾁ
34	"	66	B	98	b	162	｢	194	ﾂ
35	#	67	C	99	c	163	｣	195	ﾃ
36	$	68	D	100	d	164	､	196	ﾄ
37	%	69	E	101	e	165	･	197	ﾅ
38	&	70	F	102	f	166	ｦ	198	ﾆ
39	'	71	G	103	g	167	ｧ	199	ﾇ
40	(	72	H	104	h	168	ｨ	200	ﾈ
41	)	73	I	105	i	169	ｩ	201	ﾉ
42	*	74	J	106	j	170	ｪ	202	ﾊ
43	+	75	K	107	k	171	ｫ	203	ﾋ
44	,	76	L	108	l	172	ｬ	204	ﾌ
45	-	77	M	109	m	173	ｭ	205	ﾍ
46	.	78	N	110	n	174	ｮ	206	ﾎ
47	/	79	O	111	o	175	ｯ	207	ﾏ
48	0	80	P	112	p	176	ｰ	208	ﾐ
49	1	81	Q	113	q	177	ｱ	209	ﾑ
50	2	82	R	114	r	178	ｲ	210	ﾒ
51	3	83	S	115	s	179	ｳ	211	ﾓ
52	4	84	T	116	t	180	ｴ	212	ﾔ
53	5	85	U	117	u	181	ｵ	213	ﾕ
54	6	86	V	118	v	182	ｶ	214	ﾖ
55	7	87	W	119	w	183	ｷ	215	ﾗ
56	8	88	X	120	x	184	ｸ	216	ﾘ
57	9	89	Y	121	y	185	ｹ	217	ﾙ
58	:	90	Z	122	z	186	ｺ	218	ﾚ
59	;	91	[	123	{	187	ｻ	219	ﾛ
60	<	92	¥	124	¦	188	ｼ	220	ﾜ
61	=	93	]	125	}	189	ｽ	221	ﾝ
62	>	94	^	126	~	190	ｾ	222	ﾞ
63	?	95	_	127		191	ｿ	223	ﾟ

文字型と整数型の関係

Cにおいては，文字型と整数型がほとんど区別なく扱われます．すなわち，文字データと，それを表す整数は，どちらにでも自由に解釈して処理することができます．たとえば（JIS コードの場合）

Aのコードは65
Bのコードは66　　　（前のページの表より）

ですから，

'A'+1

は

65+1

と同じことになり，したがってその答は

数値として解釈すれば66
文字として解釈すれば'B'

ということになります．実際にやってみましょうか？

```
// example 4.9
#include <iostream.h>

main( )
{
    cout << char('A'+1);
}
```

【実行例】

```
B
```

プログラム例4.9――アルファベット

文字型は「整数型と同じ扱い」なのでfor文の制御変数として使うことができます．これを利用して

A～Z　a～z　ア～ン　0～9

の文字を全部表示してみましょう．

```
// example 4.10
#include <iostream.h>

main( )
{
  char a;
  for (a='a'; a<='z'; ++a) cout << a;
  cout << '¥n';
  for (a='A'; a<='Z'; ++a) cout << a;
  cout << '¥n';
  for (a='ｱ'; a<='ﾝ'; ++a) cout << a;
  cout << '¥n';
  for (a='0'; a<='9'; ++a) cout << a;
  cout << '¥n';
}
```

【実行例】

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ｱｲｳｴｵｶｷｸｹｺｻｼｽｾｿﾀﾁﾂﾃﾄﾅﾆﾇﾈﾉﾊﾋﾌﾍﾎﾏﾐﾑﾒﾓﾔﾕﾖﾗﾘﾙﾚﾛﾜﾝ
0123456789
```

プログラム例4.10——コード表を作る

「文字型と整数型はほとんど同じ」ですから，0～255の整数を文字として出力すれば文字が表示されます．ただし，0から31までは表示できない制御文字なので省いた方がよく，また

128～159　　　224～255

の範囲はMS-DOSの上では「2バイト・コード（漢字）」の一部と解釈されて画面の混乱の原因になりますので使わない方がよいでしょう．上記の範囲を除いて，整数値と文字の対応表を作ってみたのが次のプログラムです．

```
// example 4.11
#include <iostream.h>
#include <iomanip.h>
#define TAB '¥t'

main( )
{

  int i,j;
  for (i=0; i<32; ++i) {
    for (j=32; j<128; j+=32) {
      cout << setw(4) << i+j;
      cout <<' '<<char(i+j)<<TAB;
    }
    for (j=160; j<224; j+=32) {
      cout << setw(4) << i+j;
      cout <<' '<<char(i+j)<<TAB;
    }
    cout << '¥n';
  }
}
```

113ページのコード表は，このプログラムで作成しました．皆さんもこのプログラムをご自分の計算機で走らせてみて下さい．機種によって少し違う結果が出ることがあります．外国系の機種だと92番が逆斜線＼になるでしょう．

プログラム例4.11――大文字→小文字変換

113ページのコード表を見ると

(大文字のコード) + 32 = (小文字のコード)

になっています．これを用いて「入力された大文字を小文字に変換して表示するプログラム」を作ってみましょう．

```
// example 4.12
#include <iostream.h>

main( )
{
  char oomoji;
  cout <<"大 文 字 を 1 字 入 れ て 下 さ い ";
  cin >> oomoji;
  cout << char(oomoji+32);
}
```

【実行例】

```
大文字を１字入れて下さいG
g
```

問7.1　上のプログラムに「入力された文字が確かに大文字であるか否か」の判定を加え，大文字でない場合には変換しないように改造して下さい．

問7.2　「32を加える」かわりに「32に相当するビットを付加する」

`oomoji | 32`

という形でも大文字→小文字変換ができる筈です．やってみましょう．

4.4　文字列は文字型の配列として扱う

文字を並べたものが文字列ですから，文字列は「文字型の配列」として扱うことができます．文字列変数を

　　char 変数名 [文字数プラス 1] ;

で宣言するのはそのためです．文字列 a の第 i 番めの文字は a[i-1] という形で参照することができます．添字は 0 から始まるので，先頭は a[0] です．文字列の最後には終端記号を付けます．終端記号のコード番号は 0 です．

（例）

実際には次のようにコード（番号）の形で表されています．

206	221	188	222	194	202	190	178	195	221	197	216	0

プログラム例4.12――ラフォーレ

文字列を入力し，先頭から1字ずつ取り出して表示してみましょう．

```
// example 4.13
#include <iostream.h>

main( )
{
  char a[100];
  int i=0;
  cout << "なにか文字列を入れて下さい ¥n";
  cin >> a;
  while (a[i]!=0)
    cout<<"a["<<i<<"]="<<a[i++]<<'¥n';
}
```

【解説】 文字列の終りには必ず終端コード（番号0）が付いていますので，それが来るまで`while(a[i]!=0)`で反復しています*.

【実行例】

```
なにか文字列を入れて下さい        なにか文字列を入れて下さい
LAFORET                           ヨコハマ
a[1]=L                            a[1]=ヨ
a[2]=A                            a[2]=コ
a[3]=F                            a[3]=ハ
a[4]=O                            a[4]=マ
a[5]=R
a[6]=E
a[7]=T
```

* カーニハン&リッチーの流儀で書けばもっと短くなりますが，ここでは初心者への配慮により，わかり易いスタイルを示しておきます．

4.5　文字列の代入と比較

一般に a, b を配列の名前とするとき

```
a=b;
```

によって配列 b の内容を配列 a に代入したり

```
a<b     a>b
```

などによって配列 a, b の文字列を比較したりすることはできません．文字列は「文字型の配列」ですから，上記の理由により，全体としての代入や比較はできません．文字列を代入しようと思ったら，たとえば

```
for(i=0;b[i]!=0;++i)
    a[i]=b[i];
```

というようにして1字ずつ転記する必要があり，また比較も，先頭から1文字ずつ比較しなければなりません．（比較は，110ページ参照．）

でも，代入とか比較というような基本的な操作を，いちいち for 文を使って書くのはめんどうですね．そこで，C や C++ の処理系では，文字列の代入と比較のための関数を用意してあります．これらの関数を使用する際には，

```
#include <string.h>
```

が必要です．

代入　`strcpy(` 代入先 `,` 代入元 `);`

代入文と同様，代入先を先に書くことに注意！

比較　`strcmp(` 文字列$_1$ `,` 文字列$_2$ `);`

文字列$_1$ の方が小さければ負，大きければ正，等しければ 0 になる．

プログラム例4.13——文字列コピー

strcpy によって文字列のコピーをしてみましょう.

【実行例】

```
文字列を入れて下さい
GAO
いまの文字列は配列 a に入りました
文字列 a を b にコピーします
文字列 b を表示します
GAO
```

```
// EXAMPLE 4.14
#include <iostream.h>
#include <string.h>

main( )
{
  char a[100],b[100];
  cout<<"文字列を入れて下さい ¥n";
  cin >>a;
  cout<<"いまの文字列は配列 a に入りました¥n";
  cout<<"文字列 a を b にコピーします¥n";
  strcpy(b,a);              // 文字列のコピー
  cout<<"文字列 b を表示します¥n";
  cout<<b<<'¥n';
}
```

プログラム例4.14——文字列の比較

strcmp によって文字列の比較をしてみましょう.

```
// example 4.15
#include <iostream.h>
#include <string.h>

main( )
{
  char a[100],b[100];
  int daishou;
  cout << " a = ";  cin >> a;
  cout << " b = ";  cin >> b;
  daishou=strcmp(a,b);
  if (daishou< 0) cout << "a<b ¥n";
  if (daishou==0) cout << "a=b ¥n";
  if (daishou> 0) cout << "a>b ¥n";
}
```

【実行例】

```
 a = デカ
 b = チビ
a>b
```

```
 a = PARCO
 b = parco
a<b
```

プログラム例4.15——文字列のソート

文字列をABC順（アイウエオ順）に並べかえるプログラムを作ってみましょう.

このプログラムを書くには，研究すべき問題がいろいろあります．まず，「文字列の配列」をどうやって表すか，ということが問題になります．これは右の例のように添字を二つ付けて

a[i][j]

第1添字は文字列の番号　第2添字は文字位置

というように使うのがよく，そのようにすれば，

```
cin  >> a[i];
cout << a[i];
```

によって a[i] の入出力を行うことができます.

ソーティングの方法としては，簡単のため，例4.3に示した「バブル・ソート」という算法を用いることにします．それには「二つの文字列の交換（入れかえ）」が必要になりますが strcpy を3回使って実現することにしました.

【右のプログラムの実行例】

```
n=5                     並べ換えた結果
a[0]=タナカ              a[0]= コウダ゛ツ
a[1]=ソウマ              a[1]= サノ
a[2]=タカハシ            a[2]= ソウマ
a[3]=コウダ゛ツ          a[3]= タカハシ
a[4]=サノ                a[4]= タナカ
(右の列に続く)
```

【プログラム例】

```
// example 4.16
#include <iostream.h>
#include <string.h>

// ABC(アイウエオ) 順 に 並 べ 換 え る
main()
{
  const int M=10;    //文 字 列 の 長 さ
  const int N=30;    //文 字 列 の 個 数

  char a[N][M],w[M];
  int i,n,j,made;
  // 入 力
  cout<<"n=";   cin>>n;
  for (i=0; i<n; ++i) {
    cout<<"a["<<i<<"]=";
    cin>>a[i];
  }
  // ソ ー ト
  for (made=n-2; made>=0; --made) {
    for (j=0; j<=made; ++j) {
      if (strcmp(a[j],a[j+1])>0) {
        strcpy(w,a[j]);
        strcpy(a[j],a[j+1]);
        strcpy(a[j+1],w);
      }
    }
  }
  // 出 力
  cout << "¥n 並 べ 換 え た 結 果  ¥n";
  for (i=0; i<n; ++i)
    cout << "a[" << i << "]= "
              << a[i] << '¥n';
}
```

4.6　1文字単位の入出力

これは1文字単位の入出力のための関数で

　　getchar()

は1文字読み込みを表し,

　　putchar(文字型変数名);

によって1文字を出力することができます.

プログラム例4.16——タヌキコトバ

ピリオドが来るまで文字列を読み込み，そのなかの「タ」の字だけを除いて残りをそのまま出力するプログラムを作ってみましょう.

```
// example 4.17
#include <iostream.h>
#include <stdio.h>

main( )
{
  char c=' ';
  int i=0;
  cout<<"カナ文字列を入れて下さい¥n";
  cout<<"最後は英字のピリオドを";
  cout<<      "入れて下さい ¥n";
  while(c!='.') {
    c=getchar();
    if (c!='ﾀ') putchar(c);
  }
}
```

getchar( )の使い方は

```
int c;
```

のように int 型で宣言した変数に

```
c=getchar();
```

として使用します．文字型なのになぜ int 型を使うかというと，ファイルの終わりにきたという印として getchar( )は int 型の EOF という，256種の文字コード以外の値を返すからです．

このプログラムの1行目には，

```
#include <stdio.h>
```

がありますが，getchar( )，putchar( )，EOF などを使用するときは，必ず入れます．これは，getchar( )などが，マクロとして stdio.h というファイルの中で定義されているからです．

もしこの行を入れないと，リンク時に「関数や変数が未定義」というエラーが出てしまいます．

【実行例】

```
ｶﾀﾀﾀｷ
ｶｷ
ｳﾀｳﾀｳ
ｳｳｳ
ﾀｯﾌﾟﾀﾞﾝｽ
ｯﾌﾟﾞﾝｽ
```

4.7 補　足

文字列データの扱い方に関する以上の説明は，最も基本的な書き方で，これに関してはCにおける書き方と全く同じです．

それに対し，C++で拡張された新しい機能を使えば，

文字列の代入（コピー）を代入文の形で書く．

（例）　`a=b;`

文字列の比較を比較演算子で表す．

（例）　`if (a==b)` …

文字列の連接*を＋記号で表す．

（例）　`c=a+b;`

というような快適な書き方ができます．ただし，そのためには「文字列をそのような方式で扱うためのクラス」を用意しなければなりません．

残念ながら，本書の執筆時点では，これがまだ標準化されておらず，誰にも（どこにも）通用するような形で説明することができないので，詳しい説明は省略しますが，たとえばMicrosoft C++の場合には，プログラムの冒頭に

```
#include <CString.h>
```

を書き，文字列を扱う変数（いまの例の a, b, c など）を

```
CString a,b,c;
```

というような形で宣言しておけば，＝による代入，==による比較，＋による連接などを行なうことができます．

---

* つないで一つの文字列にすること

5

関数の書き方と使い方

長い複雑なプログラムを書くときには，プログラム全体をいくつかの小さな部分に分けて，

小さなプログラム①	全体で一つのプログラム
小さなプログラム②	
小さなプログラム③	
・・・・・・	

の形で書くのが普通です．

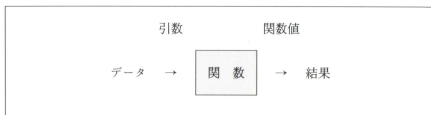
この章では，そのような書き方の基礎となる「関数」について説明します．

5・1 基本的な書き方

関数というのは，何かデータを受け取り，それについて計算をして，結果を返すものです．そのための

データの入り口が「引数」

結果の返却口が「関数値」

です．

関数の典型的な例として **sqrt(x)** を考えてみましょう．これは平方根の計算をする（関数という形式の）プログラムで，呼び出されると引数 **x** をもとに，その平方根を計算し結果を「関数値」という形で返します．

2 → √ → 1.41421356…

この章では，まずそのような（普通の）関数の書き方について説明します．ところで，CやC++においては，他のプログラミング言語（FORTRAN, BASIC, PASCAL など）における

サブルーティン，手続き（procedure）

に相当するものも「関数という形式で書く」という（ちょっと風変わりな）文法になっています．この章の後半では，そのような（拡張された）使い方について説明します．

関数（の定義）は一般に次の形式で書きます．

```
結果の型名　関数名 ( 引数とその型宣言の列 )
{
    処理手続きの記述

    return 関数値 ;
}
```

(例)
```
double f(double x,double y)
{
  double t;
  t=2.0*x+3.0*y;
  return t;
}
```

◆ 「結果の型名」を省略すると「結果は整数型」と解釈されます．

◆ 関数名のつけかたの規則は変数名のつけかたと同じです．

◆ return の右には，その関数の値として引き渡すべき値（いわば「答」）を書きます．なお，値を返す必要のないときは return を省略できます．( ) を付ける義務はありませんが，見易くするため

```
return ( 関数値 );
```

という書き方がよく用いられます．

関数を使う（呼び出す，引用する）側の書きかたは，普通の数式と同様に，たとえば

```
y=f(x)+g(x) ;
```

といったぐあいに書けばよいわけですが，次の点に注意して下さい．

◆　使う側でも原則として「関数の型の宣言」が必要です．書き方は

　　型名　関数名 (引数の型の列) ;

◆　ただし，関数の定義を先に書き，そのあとにその関数を使う関数を書き，最後にメインプログラムを書くと「関数の型の宣言」は省略できます．

```
関数の定義を先に書く

main()
{
  メイン・プログラムの記述
}
```

プログラム例 5.1 ―― 1 次関数

実数型の引数 x について

$$f(x) = 2x + 3$$

の値を計算して倍精度実数型で値を返す関数 $f(x)$ は次のように書きます．ついでにテスト用の簡単なメイン・プログラムを付けてみました．

```
// example 5.1a
#include <iostream.h>

double f(double x)
{
  return 2.0*x+3.0;
}

main( )
{
  double x,y;
  cout << "x=";       cin >> x;
  y=f(x);
  cout << "f(x)=" << y << '¥n';
}
```

【実行例】

```
x=5.0
13
```

プログラム例5.2――最大公約数

二つの正の整数 m, n の最大公約数を計算するプログラム（関数）

gcd(m,n)

を作ってみましょう．

```
// example 5.2
#include <iostream.h>

int gcd(int m,int n)
{
   int p,q,r;
   if (m<n) { q=m; p=n; }
      else  { q=n; p=m; }
   while (q>0) {
     r=p%q;
     p=q;
     q=r;
   }
   return(p);
}

main( )
{
   int m,n;
   cout << "m=";  cin >> m;
   cout << "n=";  cin >> n;
   cout << "gcd=" << gcd(m,n) << '¥n';
}
```

【実行例】

```
m=24
n=30
gcd=6
```

【解説】 このプログラムは「ユークリッドの互除法」という算法を用いています．m と n の内，大きい方を p，小さい方を q として開始し，

p を q で割った余りを r とする．　新 $p = q$　新 $q = r$

という手続きを $r = 0$ になるまで反復したとき，最後の除数が最大公約数になります．

5.2 引数についての決まり

関数というのは，もともと

$$y = 2x + 1$$

$$y = \sin x$$

$$y = f(x)$$

のようなものですから，

引数として「値」を受け取り，

それをもとに処理をし，

結果を「関数値」として返す，

という使いかたが基本的です．図で書くなら，

値 → 関　数 → 値

（引数）　　　　（関数値）

です．したがって

引数は「値」を「関数に引き渡す」ための窓口

であり，原則として一方通行です．

Cの場合，この原則どおりに文法ができておりますので，たとえば$f(x)$という関数の中で，その仮引数に **x=… ;** と代入しても，もとのプログラム（呼び出した側）の変数の値は変りません*．

プログラム例5.3――愛は変らず

関数$f(i)$の中でiに5を代入し，メインプログラムの実引数を表示してみましょう．

```
// example 5.3
#include <iostream.h>

int f(int i)
{
  i=5;
    cout << "¥n i に 5 を代入 ¥n";
  return 2*i+3;
}

main( )
{
  int i,j;
  i=3;              // i に 3 を入れる
  cout <<"i="<<i;  // 確認
  j=f(i);           // f (i) を呼び出す
  cout <<"i="<<i;  // 変っているかな？
}
```

【実行例】

```
i=3
 i に 5 を代入
i=3
```

iに5を代入したのに，メイン・プログラムの **i** は **3** のままです．

* FORTRANだと変ります．やってみて下さい．

引数が一方通行になっているのは，多くの場合，安全で良いことですが，場合によっては

実引数（呼び出し側の引数）の値を関数側で書きかえたい

引数を介して結果を返したい

ということがあります．

たとえば，結果として二つ以上の値を返したい場合があります．関数値として戻せるのは一つだけですから，残りは引数を介して返さなければなりません．

`swap(a,b)`　a と b の値を入れかえる

`sort(x)`　配列 x の内容を小さい順に並べかえる

というような関数を書こうとすれば，当然「引数の値を書きかえる」ことが必要になります．

そういう場合，他の言語なら，関数でない形（サブルーティン，手続き等）を使うのですが，C には関数しかないので，関数の文法の目をごまかして（専門用語で「副作用――side effect」という手を使って）実現しなければなりません．

その抜け道として最もよく使われるのが「参照渡し」という書き方です．引数として「もとの変数の番地」を渡せばそこに書き込んでしまうことができます．実例をいくつか掲げておきますので，よく研究し，自分でもそういうプログラムを書けるように練習をして下さい．

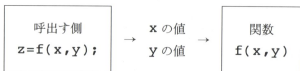
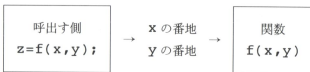
値渡しと参照渡し

一般に，関数を呼び出すときの引数の渡し方には

値渡し

参照渡し

という二つの方式があります．

値渡しの場合には，関数を呼出すとき，実引数（呼出す側の引数）の値そのもの（正確にいえば，値のコピー）を関数に渡します*.

それに対し，参照渡しの場合には，関数を呼出すときに，実引数として書かれている変数の記憶場所（番地）を関数に渡します**.

呼出された関数の側では，引数の値が必要になった時点で，教えてもらった番地（データが入っている場所）に値を読みに行きます．

* 本書でこれまで説明してきたのは「値渡し」の方式です．

** 他のプログラミング言語（たとえば FORTRAN など）では「原則として参照渡し」という規則になっているものが多いようです．

両方式にはそれぞれ一長一短があります．明快で初心者に理解しやすいのは値渡しの方でしょう．しかし，値渡しは一方通行ですから，引数を介して答を返すことができません．135ページで説明したように，これでは二つ以上の答を返したい場合に困ります．

C++では「特に指定しなければ値渡しになる」という規則になっています．しかし，必要に応じて参照渡しを指定できるようになっています．

> 参照渡しにしたいときには，関数の宣言における「引数の型」の語尾に & 印を付けます*.

(例) 値渡し

```
int f(int x)
{
  return 2*x+3;
}
main()
{
  int i,j;
  i=5;
  j=f(i);
  cout << j;
}
```

参照渡し

```
int f(int& x)
{
  return 2*x+3;
}
main()
{
  int i,j;
  i=5;
  j=f(i);
  cout << j;
}
```

* Cでは，これと似たような目的で，実引数の頭に & を付けますが，文法上は全く違う意味ですので，混同しないように注意して下さい．

プログラム例5.4――内容交換

変数 a, b の内容をとりかえる関数 swap を書いてみましょう．

```
// example 5.4
#include <iostream.h>

void swap(int& a,int& b)
{
  int c;
  c=a;      // a と b を入れ替える
  a=b;
  b=c;
}

main( )
{
  int a,b;
  cout << "a=";  cin >> a;
  cout << "b=";  cin >> b;
  cout << "入れ替えます¥n";
  swap(a,b);
  cout << "a=" << a << '¥n';
  cout << "b=" << b << '¥n';
}
```

【実行例】

```
a=298
b=808
入れ替えます
a=808
b=298
```

プログラム例5.5——座標変換

二つの実数値 r, θ を受け取り，$x = r\cos\theta, y = r\sin\theta$ を計算して返す関数（極座標から直角座標に変換するプログラム）を書いてみましょう．

```
// example 5.5
#include <iostream.h>
#include <math.h>

void ptoc(
  double r,
  double theta,
  double& xx,
  double& yy)
{
  xx=r*cos(theta);
  yy=r*sin(theta);
}

main( )
{
  const double dtor=3.141593/180.0;
  double r,theta,t,x,y;
  // 極座標を読み込む
  cout << "r=";   cin >> r;
  cout << "theta=";   cin >> theta;
  // 角度の単位をラジアンに換算
  theta*=dtor;
  // 極座標を直行座標に変換
  ptoc(r,theta,x,y);
  // 結果を表示
  cout << "x=" << x << '¥n';
  cout << "y=" << y << '¥n';
}
```

【実行例】

```
r=2
theta=60
x=1
y=1.73205
```

【解説】 r と t は「値を受け取るだけ」なので普通の書きかたでよく，x と y は「値を返してもらう」必要があるので，ポインタを渡して書き込んでもらう方式にしてあります．なお，`dtor=3.141593/180.0` の式は角度の単位の度からラジアンへの変換のためです．

5.3 配列と文字列の渡し方

前節で，

C++ では原則として値渡しになる

参照渡しにしたい場合は，仮引数の型宣言の語尾に & を付ける

と説明しましたが，これには少し例外があります．それは，

> 引数が
> 　配列名
> 　文字列名
> の場合には，& を付けなくても（自動的に）参照渡しになる

という規則です．その理由は，値渡しにすると大量のデータをコピーしなければならず，時間的にもメモリー容量の点でも損になるからでしょう．

そのため便利なこともありますが，注意を要する点もあります．たとえば消去法で連立 1 次方程式を解く関数を呼び出すと，係数行列や右辺の内容が全部書き変えられてしまう可能性があります．

プログラム例5.6——合 計

配列 a に入っている一組のデータ $a_0, a_1, \cdots, a_{n-1}$（全部で n 個）の合計を求めるプログラムは次のように書きます．

```
// example 5.6
#include <iostream.h>

double sum(double a[],int n)
{
  double s=0.0;
  for (int i=0 ; i<n; ++i) {
     s+=a[i];
  }
  return s;
}

main( )
{
  int n;
  // 配 列 a の 適 応 的 宣 言
  cout << "n=";  cin >> n;
  double* a=new double[n];
  // デ ー タ を 読 み 込 む
  for (int i=0; i<n; ++i) {
    cout << "a[" << i << "]=";
    cin >> a[i];
  }
  // 関 数 を 呼 び 出 し て 計 算
  double k=sum(a,n);
  // 出 力
  cout << "¥n合 計 " << k << '¥n';
}
```

【実行例】

```
n=5
a[0]=1
a[1]=2
a[2]=3
a[3]=4
a[4]=5

合計15
```

プログラム例5.7——小さい順

整数 $a_0, a_1, \cdots, a_{n-1}$ を小さい順に並べかえるプログラムを関数の形で作ってみます.

```
// example 5.7
#include <iostream.h>

void sort(int a[],int n)
{
  for (int m=n-2; m>=0; --m) {
    for (int j=0; j<=m; ++j) {
      if (a[j]>a[j+1]) {
        int w=a[j];
        a[j]=a[j+1];
        a[j+1]=w;
      }
    }
  }
}

main( )
{
  int n;
  cout << "n=";  cin >> n;
  int* a=new int[n];
  // データの読み込み
  for (int i=0; i<n; ++i) {
     cout << "a[" << i << "]=";
     cin >> a[i];
  }
  // 関数 sort を呼び出す
  cout << " *** ソートします *** ¥n";
  sort(a,n);
  // 出力
  for (i=0; i<n; ++i) {
     cout << "a[" << i << "]=";
     cout << a[i] << '¥n';
  }
}
```

【実行例】

```
n=5
a[0]=37
a[1]=45
a[2]=12
a[3]=60
a[4]=25
 *** ソートします ***
a[0]=12
a[1]=25
a[2]=37
a[3]=45
a[4]=60
```

プログラム例5.8——小文字→大文字変換

与えられた文字列の中の英字の小文字を大文字に変換して返す関数を作ってみましょう．

```
// example 5.8
#include <iostream.h>

void cap(char komoji[],char oomoji[])
{
  char c;
  int  i=0;
  while ((c=komoji[i])!=0) {
    if (96<c && c<123)
      oomoji[i]=c-32;
    ++i;
  }
  oomoji[i]=0;    // 終端記号を追加
}

main( )
{
  char komoji[80],oomoji[80];
  cout<<"単語を小文字で入れて下さい¥n";
  cin >> komoji;
  cap(komoji,oomoji);
  cout << oomoji;
}
```

【実行例】

```
単語を小文字で入れて下さい
mips
MIPS
```

5.4　関数の中で使える変数

関数の中で使用できる変数（およびそれに準ずるもの）としては

自動変数
静的変数
大域変数

の3種類があります．この内，自動変数というのが要するに「普通の変数」で，あとの二つは特殊なのですが，この際まとめて覚えておくことにしましょう．

1.　自動変数

これは関数の計算に必要な作業場所（中間結果などを一時的に記憶しておく所）として使う変数で，その名前は宣言した関数の中だけで有効です．自動変数を実際に記憶する場所は，その関数が呼び出された時点で自動的に割りつけられ，**return** の際に抹消されます．自動変数の宣言は関数を定義するプログラムの {} 内に書きます．

2.　静的変数

これは「次に呼び出される時まで値を保持しておきたい場合に使う変数」です．自動変数だと **return** の際に値が消えてしまいますが，静的変数だと次に呼び出される時まで値を残しておいてくれます．静的変数の宣言は，関数を定義するプログラムの {} 内に，型名の前に **static** という予約語をつけて書きます（後の例参照）．静的変数でも名前はその関数の中だけで有効です．

3. 大域変数

これは，すべての関数で共通に使用できる変数で，その変数名はプログラムの全域で通用します*．大域変数の宣言はプログラムの冒頭に書きます（すべての関数の外側．main よりも前です）．

大域変数を乱用することはあまり好ましいことではありませんが，C の場合，前述のとおり引数の扱いが結構めんどうなので，引数を使うかわりに大域変数によるデータ受け渡しがよく用いられています．

また，大域変数の頭に static を付けることができます．こうすると，そのファイル内でのみ，その変数が見え，他のファイルからは見えなくなります．

```
変数の宣言                    ……[ 大域変数 ]
static 変数の宣言             ……[ ファイル内の大域変数 ]
型名  関数名 ( 引数の列 )
引数の型宣言
{
auto 変数の宣言               ……auto を省略可 [ 自動変数 ]
static 変数の宣言             ……[ 静的変数 ]
…
}
main( )
{
…
}
```

* ただし，一つのプログラムをいくつものファイルに分けて書いた場合には個々の関数の中であらためて external 宣言を行う必要があります．

プログラム例5.9——乱　数

静的変数の典型的な応用例として，乱数発生用の関数を一つお目にかけましょう．呼び出す毎にでたらめな整数が出てきます．

【解説】 ここで用いている方法は「平方採中法」といって，m 桁の乱数の2乗（$2m$ 桁になる）の中央 m 桁を「次の乱数」にするやりかたで，他の言語だと「中央 m 桁をとり出す」のに苦労しますが，Cだとシフト演算子があるので好都合です．

```
// example 5.9a
#include <iostream.h>

int rans(void)
{
  // n を静的変数として宣言し、
  // 出発値をセット
  static long n=24689;
  // n を 2 乗し、
  // 左に 2 進 8 桁シフトし、
  // 右に 2 進 1 6 桁シフトする
  n = ( (n*n)<<8 ) >> 16;
  // 整数型に変換して返す
  return int(n);
}

// メイン・プログラムの例
main( )
{
  for(int i=0; i<10; ++i)
    cout << rans() << '¥n';
}
```

【実行例】

```
21745
12042
-23380
-27437
-8539
22678
-22665
-24967
10133
7868
```

【別解】 静的変数を使う目的は「次に呼ばれる時まで値を残す」ことですから，大域変数を使っても同じ効果を出すことができます．以下に示すのは，左のプログラムをそのような形に書きかえたものです．ついでにサイコロのかわりに使えるように，1から6までの整数が同じ確率で出るように改造してみました．また，乱数のタネ（出発値）をキーボードから入力できるようにしました．出発値が適当でないと，すぐ脱線してしまいます．いろいろな値（たとえば123456など）を入れて実験してみるとよいでしょう．

```
// example 5.9b
#include <iostream.h>
#include <math.h>
// nを大域変数として宣言
long n=24689;

int rans(void)
{
  // 平方採中法による乱数生成
  n = ( (n*n)<<8 ) >> 16;
  // 1から6までの整数に変換して返す
  return int(abs(n)%6+1);
}

// メイン・プログラムの例
main( )
{
  // 乱数のタネを読み込む
  cout << " タネを入れて下さい ";
  cin  >> n;
  for(int i=0; i<10; ++i) {
    for(int j=0; j<10; ++j)
      cout << rans() << "  ";
    cout << '¥n';
  }
}
```

【実行例】

```
タネを入れて下さい 1987
3  2  6  3  3  6  4  2  6  5
3  6  6  3  6  3  1  3  6  2
4  3  3  2  3  1  5  5  5  6
3  1  3  2  5  3  3  1  1  2
6  5  4  5  5  6  2  4  4  3
```

プログラム例5.10——連立1次方程式

例4.7のプログラムを，入力，計算，出力の三つの部分に分けて書くと次のようになります．こういう場合に大域変数を使うと便利です．

```
// example 5.10
#include <iostream.h>
const int NN=30;
int n ;
double a[NN][NN];     /* 係数行列   */
double b[NN];         /* 定数項     */
double x[NN];         /* 解ベクトル */

/******* 連立1次方程式を解く *******/
void leq(void)
{
  int i,j,k;
  double p,q,s;
  for (k=1; k<n; ++k) {
    p=a[k][k];
    for (j=k+1; j<=n; ++j) {
      a[k][j]/=p;
    }
    b[k]/=p;
    for (i=k+1; i<=n; ++i) {
      q=a[i][k];
      for (j=k+1; j<=n; ++j) {
        a[i][j]-=q*a[k][j];
      }
      b[i]-=q*b[k];
    }
  }
  x[n]=b[n]/a[n][n];
  for (k=n-1; k>=1; --k) {
    s=0.0;
    for (j=k+1; j<=n; ++j) {
      s+=a[k][j]*x[j];
    }
    x[k]=b[k]-s;
  }
}
```

```
/******* 入力 *******/
void datain(void)
{
  int i,j;
  cout << " 元数nを入れて下さい¥n";
  cin >> n;
  for (i=1; i<=n; ++i) {
    for (j=1; j<=n; ++j) {
      cout<<"a["<<i<<"]["<<j<<"]=";
      cin >> a[i][j];
    }
    cout << "b[" << i << "]=";
    cin >> b[i];
  }
}

/******* 出力 *******/
void disp(void)
{
  int i;
  cout << "解 ";
  for (i=1; i<=n; ++i) {
    cout << "x[" << i << "]=";
    cout << x[i] << '¥n';
  }
}

/******* メイン プログラム *******/
main( )
{
  datain();   // データを読む
  leq();      // 計算
  disp();     // 結果を表示
}
```

【実行例】

```
元数nを入れて下さい
3
a[1][1]=6
a[1][2]=1
a[1][3]=8
b[1]=30
a[2][1]=7
a[2][2]=5
a[2][3]=3
b[2]=30
a[3][1]=2
a[3][2]=9
a[3][3]=4
b[3]=30
解x[1]=2
x[2]=2
x[3]=2
```

5.5 再帰呼出し

手続きや関数の中で，他の手続きや関数を呼び出すことができます．他のプログラミング言語，たとえばBASICやFORTRANでもできて，ごく自然に使えます．

ところでCでは，手続きや関数の中で自分自身を呼び出すという奇妙なことができます．これを再帰呼出し（recursive call）と言います．まず簡単な例で説明しましょう．

プログラム例5.11——*n*!

$n!$（nの階乗，すなわち$1 \times 2 \times 3 \times \cdots \times n$）を計算する関数$f(n)$を再帰呼出しの形で書くと次のようになります．

```
// example 5.11
#include <iostream.h>

int f(int n)
{
  int kaijou;
  if (n==0)
    kaijou=1;
  else
    kaijou=n*f(n-1);
  return kaijou;
}

main( )
{
  int m,n;
  cout << "n=";    cin >> n;
  m=f(n);
  cout << "n!=" << m << '¥n';
}
```

【実行例】

```
n=5
n!=120
```

【解説】 このプログラムは

$$n! = n \times (n-1)!$$

という漸化式を使って計算しています．この式が正しいことは

$$n! = n \times \underbrace{(n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1}_{\text{ここをまとめると } (n-1)!}$$

だから容易にわかりますね．

プログラムの中では！の記号を使えませんから，$n!$を$f(n)$で表すことにします．そうすると上の漸化式は

$$f(n) = n \times f(n-1)$$

と書くことができます．左のプログラムでは，これが`else`の所に書いてありますね．もっと簡単に

```
return((n==0)?1:n*f(n-1));
```

と書くこともできます．

一方，`if`の方ですが，nが0ならば値が1，すなわち$0! = 1$ということを書いています．$0!$を1とするのは数学の約束です．でも，これが不自然だと思ったら，`if(n==0)`のかわりに`if(n==1)`にしても結構です．

【蛇足】「自分自身を呼び出してよい」といっても

```
return(f(n));
```

では困ります．文法違反にはならないでしょうが，これでは問題の解決にならず，実行したら止まらなくなってしまいます．

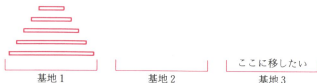
いまの例は，生まれてはじめて再帰呼出しを習う人のための最も簡単な例として示しました．このプログラムを実用に使うのはおすすめしません．$n!$ の計算は再帰呼出しなどを使わずに

```
f=1;
for(i=1;i<=n;++i)
    f*=i;
```

と書く方が速くできます．階乗の話に限っていうなら，数表を記憶させておいて使うのが最も賢明です．

一般に，再帰呼出しは時間がかかり，記憶場所もたくさん使います．もしも他に簡単な方法があれば，再帰呼出しを使わないで済ませる方がよいでしょう．

しかし，問題によっては，再帰呼出しを使わなければうまく書けないものがあります．その簡単な1例として，ハノイの塔という問題について考えてみましょう．

500円玉の上に10円玉を乗せ，その上に100円玉，その上に50円玉，1番上に1円玉を乗せると図のようになります．

最初，それを基地1に置きます．それを基地3に移せ，というのが問題です．ただし

　　1度に1枚しか移せない

　　小さいお金の上に大きいお金を乗せてはいけない

というルールがあります．そのため，一時的な置き場として基地2を使うことができます．

はじめてやる人にとっては意外にむずかしい問題です．最初は2枚でやってみて，それができたら3枚に挑戦します．4枚をやるのは，3枚を完全にマスターしてからにするのがよいと思います．

やってみて下さい．

だんだんと要領がわかってくるでしょう．2枚の場合が基本でして，

　　上の1枚（これを帽子と呼ぶことにします）を空き地に移す

　　下の1枚（これを台と呼ぶことにします）を目的地に移す

　　空き地に置いてあった帽子を目的地の台に乗せる

という手順で移せます．

何枚もあるときは，枚数をnとすると

　　上の$n-1$枚を帽子のつもりで空き地に移す

　　一番下の1枚（台）を目的地に移す

　　空き地の帽子を台に乗せる

という要領でやります．ただし$n>2$のときは帽子を一度に移せませんから，同様に分解して移すわけです．

わかりましたか？　わかってしまえば簡単なことで，3枚，4枚ぐらいはわけなくできます．もっと枚数が多くても，ただ根気よく速く正確に実行していけばよいのです．11枚やった人もいます．

しかし，やがて「こんな単純で機械的なことはコンピュータにやらせる方がよいのでは………」と考えるようになるでしょう．

再帰呼出しのできるプログラミング言語を使えば，a から b へ n 枚移すための手続き

```
utusu(a,b,n)
```

の本文を，先ほど説明した要領のとおり

```
utusu(a,c,n-1);
utusu(a,b,1);
utusu(c,b,n-1);
```

と書けますから，簡単です．ただし c は a, b でない残りの1カ所で，それが具体的に何番になるかは，ちょっとした工夫ですが

$$c = 6 - a - b$$

で算出できます（$a + b + c$ が番号の和 $1 + 2 + 3 = 6$ に等しいから）．こういう方針で書いたプログラムを右のページに示します*．問題が割合むずかしいのにプログラムは意外と簡単ですね．これが再帰呼出しの威力です．再帰呼出しのできない言語でこの問題のプログラムを書くには高度な技法と相当な手間をかけなければなりません．

* 結果をアニメ風に表示しようとすると，何円玉が今どこにあるか，という情報を扱う必要がありますので，もう少し複雑になります．

プログラム例 5.12——ハノイの塔

ハノイの塔のプログラムは次のように書くことができます.

```
// example 5.12
#include <iostream.h>

void utusu(
  int kara,        // 元
  int made,        // 移す先
  int maisuu)      // 移す枚数
{
  int nokori;
  if(maisuu==1)
    cout << kara << " から  " <<
            made << " に 1 枚移す¥n";
  else {
    nokori=6-kara-made;
    utusu(kara,nokori,maisuu-1); /* 帽子 */
    utusu(kara,made,1);          /* 台   */
    utusu(nokori,made,maisuu-1); /* 帽子 */
  }
}

main( )
{
  int n;
  cout << "枚数 n=";
  cin >> n;
  utusu(1,3,n);
}
```

【実行例】

```
枚数 n=3
1 から 3 に1枚移す
1 から 2 に1枚移す
3 から 2 に1枚移す
1 から 3 に1枚移す
2 から 1 に1枚移す
2 から 3 に1枚移す
1 から 3 に1枚移す
```

5.6 インライン展開

関数の処理内容が比較的簡単な場合，関数定義の冒頭に **inline** と指定しておくと，実行時間を短縮することができます．

【解説】 **inline** の指定がなければ，関数はそれぞれ独立したプログラムになります．関数を呼び出すということは，仕事を別の会社に外注するようなものですから，きちんとした（他人行儀の）引継き手続きをしなければなりません．また，関数という「別のプログラム」を走らせるためには（自分の仕事場を他人に貸すようなもので）やりかけの仕事を片づけ，白紙の状態にして引渡さなければなりません．そういう事情から，関数を呼び出すといろいろ余計な手間（オーバーヘッド）がかかります．

しかし **inline** と指定しておけば，関数を「呼び出す側のプログラム」の一部分として（いわば社内の組織のように）扱ってくれます．たとえば，

```
inline double f(double x) { return x*x; }
```

と定義しておけば，

```
z=f(x)+f(y);
```

を，あたかも

```
z=x*x+y*y;
```

と書いたかのように扱ってくれるのです．

---

* たとえば，

```
f(x,y)=2.0*x+3.0*y;
```

のように1行ないし数行の式で書けるとか，

```
w=a; a=b; b=w;
```

のような単純な手続き，あるいは

```
if (a>b) {x=0.0; y=1.0;}
   else  {x=1.0; y=0.0;}
```

という程度．

6

構造体とクラス

この章では，C++における最も重要な「クラス」という概念と，それに関連して「構造体」のことを説明します．どちらも，常識の世界には無い新しい概念なので，少々わかりにくいかもしれませんが，非常に便利なものなので，根気よく勉強して完全に使えるようになって下さい．

6.1　考　え　方

プログラムを書くとき，いくつかのデータを組にして（ひとまとめにして）

名前を付けて

代入（コピー）したり

演算（データ処理）したり

関数に引渡したり，結果を受取ったり

ファイルに書込んだり，読出したり

したいことがよくあります．

（例1）　分数

分母，分子

（例2）　空間の位置

x 座標，y 座標，z 座標

（例3）　日付，時間

年，月，日

時，分，秒

（例4）　行列（マトリクス）

行数，列数，個々の要素の値

（例5）　氏名，住所，電話番号

姓，名，ふりがな

郵便番号，都道府県，市，町，番地

市外局番，市内局番，番号

（例6）　取引データ

顧客番号，品名，数量，単価，金額，日付

従来のプログラミング言語（たとえばFORTRANやBASIC）では，複数個のデータをまとめて扱う手段が「配列」だけでした．配列は確かに複数個のデータをまとめて扱うことができますが，同一の型のデータでなければ，一つの配列として扱うことができません．そこで，型の違うデータでも一つにまとめて扱うことができるように，CやC++では構造体という書き方ができるようになっています．

　　配列　　型が同じでなければいけない
　　構造体　型が同じでなくてもよい

別の見方からすれば，構造体は「構造を持つデータ」を扱うための表現形式です．利用者は構造体という形式のもとで，目的に応じて自由にデータ構造を定義し，いろいろな処理を行なうことができます．

これをもっと便利に，使いやすくしたのが「クラス」です．クラスはデータ構造を定義するだけでなく，そのデータ構造に関する基本的な操作（処理）も一緒に定義し，簡潔な形で表現できるようにします．

たとえば，分数というクラスを作れば，単に分母と分子をまとめて扱うだけでなく，分数の

　　約分，通分，加減乗除，大小比較，入出力

など，基本的な操作一式を「クラスのメンバー関数」として定義し，

　　a, b, c は分数である

と宣言して，

```
c=a+b;
```

などと書けるようになるのです．

6.2 構造体の書き方

構造体というのは，たとえば

氏名		文字列型
住所		文字列型
年令		整数型
体重		実数型
血液型		列挙型

のような，「一定の形式で，いくつかの項目を組にしたもの」のことをいいます．C では，これについて，

名前をつけることができます．

代入ができます（全部の項目をまとめて移せる）*.

関数に引き渡すことができます（同上）*.

配列を作ることができます．

また，構造体の中の個々の項目を，

構造体の変数名．項目の名前

の形，たとえば

a の名前　（a.namae で表す）

b の年令　（b.nenrei で表す）

のような形で引用して，普通の変数と同じように代入したり，演算したり，入出力したりすることができます．

* ただし Turbo C などの最近の処理系に限り可能で，今でもできない処理系がありますので注意して下さい．

宣言の書き方　まず

```
struct 構造体の名前 {
        型名            項目名 ;
        型名            項目名 ;
                ……
        型名            項目名 ;
};
```

の形でデータの構成を書き，型の宣言をします．

各「項目の型」は，int, float, char などと書くわけですが，

　文字列型の場合は　　char [最大字数 + 1]

　配列の場合は　　　　型名　配列名 [寸法] ;

と指定します．

これで「構造体の宣言」ができましたが，それは型を定義しただけですから，まだデータを扱うことはできません．そこで

```
struct 構造体の名前 変数名 ;
```

によって，その変数名が特定の形式の構造体であることを宣言します．これを実体の宣言といいます．

プログラム例6.1――時，分，秒の引き算

時刻を表す

時　分　秒

の三つのデータを組にして構造体として扱い，それを用いて経過時間を計算するプログラムを作ってみましょう．

【解説】 構造体の型の宣言は

```
struct jikan {
    int ji;
    int hun;
    int byou;
};
```

となります．CやC++では普通，} 記号のあとには ; を付けないでよいのですが，struct の宣言の場合は ; が必要ですので気をつけて下さい．構造体の定義は，普通，全部の関数で共通に使いますので，大域変数の定義と同様，プログラムの冒頭（main より前）に書きます．構造体の各項目を参照するには，前述のように

構造体を表す変数名 . 項目名

と書きます．

【プログラム例】

```
// example 6.1a
#include <iostream.h>

// 構造体の宣言
struct jikan {
  int ji;
  int hun;
  int byou;
};

// メインプログラムの例
main( )
{
  struct jikan a,b,c;
  int hour,min,sec;
  // テストデータ作成
  a.ji  =7;
  a.hun =31;
  a.byou=31;
  b.ji  =12;
  b.hun =3;
  b.byou=45;
  // 経過時間の計算
  hour=b.ji-a.ji;
  min =b.hun-a.hun;
  sec =b.byou-a.byou;
  if (sec<0) {
    sec+=60;
    min-=1;
  }
  if (min<0) {
    min+=60;
    hour-=1;
  }
  c.ji=hour;
  c.hun=min;
  c.byou=sec;
  // 結果の表示
  cout << " 経過時間 "
       << c.ji << " 時間 "
       << c.hun << " 分 "
       << c.byou << " 秒 ¥n";
}
```

【実行例】 経過時間 4 時間 32 分 14 秒

6.3　関数への渡し方

構造体の形のデータを関数に渡したり，関数から返してもらったりするには，136ページで説明した「参照渡し」を使うのが普通です．したがって

> 引数として構造体を書くときは
> ◆　実引数（呼び出し側）は普通の形で書く．
> ◆　仮引数（関数側）およびプロトタイプの引数欄には
> 　　構造体名 &　変数名
> とするのが原則です．

関数値として構造体を返すこともできます．それには関数の宣言を

構造体の名前　関数名 (仮引数)

で始め，本文に

return　構造体の変数名 ;

を書きます．

【プログラム例6.1別解】

```
// example 6.1b
#include <iostream.h>

// メインプログラムの例
struct jikan {
  int ji;
  int hun;
  int byou;
};

// 時間差を計算する関数
jikan jikansa(jikan& aa,jikan& bb)
{
  jikan  cc;
  int    j,h,b;
  j=(bb.ji)-(aa.ji);
  h=(bb.hun)-(aa.hun);
  b=(bb.byou)-(aa.byou);
```

```
  if (b<0) {
    b+=60;
    h-=1;
  }
  if (h<0) {
    h+=60;
    j-=1;
  }
  cc.ji=j;
  cc.hun=h;
  cc.byou=b;
  return cc;
}

// 時間を読み込む関数
jikan yomikomi()
{
  jikan aa;
  cin >> aa.ji >> aa.hun >> aa.byou;
  return aa;
}

// 時間を表示する関数
void hyouji(jikan& aa)
{
  cout << aa.ji << " 時間 "
       << aa.hun << " 分 "
       << aa.byou << " 秒 ¥n";
}

// メインプログラムの例
main()
{
  jikan a,b,c;
  int ta,tb,tc,r;
  cout << "始めの時間を入れて下さい ¥n";
  a=yomikomi();
  cout << "終りの時間を入れて下さい ¥n";
  b=yomikomi();
  c=jikansa(a,b);
  cout << "経過時間は ¥n";
  hyouji(c);
}
```

【実行例】

```
始めの時間を入れて下さい
1 2 3
終りの時間を入れて下さい
5 0 0
経過時間は
3 時間 57 分 57 秒
```

プログラム例6.2——分　数

分数の入出力と加減乗除のプログラム・パッケージを作ってみましょう．

```
// example 6.2
#include <iostream.h>

// 構造体の定義
struct bunsuu {
  long bunsi;
  long bunbo;
};

// 入力促進用に c を表示して aを読込む
void yomikomi(char c[],bunsuu& aa)
{
  cout << c << " の分子を入れて下さい¥n";
  cin >> aa.bunsi;
  cout << c << " の分母を入れて下さい¥n";
  cin >> aa.bunbo;
}

// 分数 aa と bb の和を計算
bunsuu wa(bunsuu& aa,bunsuu& bb)
{
  bunsuu cc;
  cc.bunsi=(aa.bunsi)*(bb.bunbo)+
            (aa.bunbo)*(bb.bunsi);
  cc.bunbo=(aa.bunbo)*(bb.bunbo);
  return cc;
}

// 分数 aa と bb の差を計算
bunsuu sa(bunsuu& aa,bunsuu& bb)
{
  bunsuu cc;
  cc.bunsi=(aa.bunsi)*(bb.bunbo)-
            (aa.bunbo)*(bb.bunsi);
  cc.bunbo=(aa.bunbo)*(bb.bunbo);
  return cc;
}
```

```
// 分 数 aa と bb の 積 を 計 算
bunsuu seki(bunsuu& aa,bunsuu& bb)
{
  bunsuu cc;
  cc.bunsi=(aa.bunsi)*(bb.bunsi);
  cc.bunbo=(aa.bunbo)*(bb.bunbo);
  return cc;
}

// 分 数 aa と bb の 商 を 計 算
bunsuu shou(bunsuu& aa,bunsuu& bb)
{
  bunsuu cc;
  cc.bunsi=(aa.bunsi)*(bb.bunbo);
  cc.bunbo=(aa.bunbo)*(bb.bunsi);
  return cc;
}

// 見 出 し  s を 付 け て  aa を 出 力
void hyouji(char s[],bunsuu& aa)
{
  cout << s << '¥t';
  cout << aa.bunsi << " / "
       << aa.bunbo << '¥n';
}

// メ イ ン プ ロ グ ラ ム の 例
main( )
{
  bunsuu a,b;
  yomikomi("a",a);
  yomikomi("b",b);
  hyouji("a",a);
  hyouji("b",b);
  hyouji("a+b",wa(a,b));
  hyouji("a-b",sa(a,b));
  hyouji("a*b",seki(a,b));
  hyouji("a/b",shou(a,b));
}
```

【実行例】

```
a の分子を入れて下さい
1
a の分母を入れて下さい
2
b の分子を入れて下さい
1
b の分母を入れて下さい
3
a       1 / 2
b       1 / 3
a+b     5 / 6
a-b     1 / 6
a*b     1 / 6
a/b     3 / 2
```

6.4 構造体の配列

構造体の配列を用いることもできます．文法は常識的に想像されるとおりで，宣言の書き方は

　　struct　構造体の名前　配列名［寸法］;

参照のしかたは

　　配列名［添字］　　　　　……構造体としての参照
　　配列名［添字］.項目名　　……個々の項目の参照

です．関数に配列全体を引渡すときは引数として配列名を書きますが，特定の配列要素を渡す場合には

　　&配列名［添字］

という書き方もできます．

プログラム例6.3――分数のソート

n 個の分数 $a_0, a_1, \cdots a_{n-1}$ を小さい順に並べかえるプログラムを構造体を使って書いてみましょう．

プログラム例6.2と同じ「bunsuu」という構造体を使うことにします．ソーティングのアルゴリズムとしては，簡単なため，バブル・ソートを用いることにすると，二つの構造体の中身の交換（swap）が必要になりますから，bswap という関数を作って利用します．

```
// example 6.3
#include <iostream.h>
#include <iomanip.h>

struct bunsuu {
  long bunsi;
  long bunbo;
};

// 分数の入れ替え
void bswap(bunsuu& a,bunsuu& b)
{
  bunsuu w;
  w=a;
  a=b;
  b=w;
}

// 分数列の表示
void hyouji(bunsuu a[],int n)
{
  int i;
  for (i=0; i<n; ++i)
    cout << setw(4) << a[i].bunsi;
  cout << '¥n';
  for (i=0; i<n; ++i)
    cout << " ---";
  cout << '¥n';
  for (i=0; i<n; ++i)
    cout << setw(4) << a[i].bunbo;
  cout << '¥n';
}
```

```
// メインプログラムの例
main( )
{
  int i,n,k,made,s1,s2;
  bunsuu* a;
  /** 入力 **/
  cout << "n=";  cin >> n;
  a=new bunsuu[n];
  for (i=0; i<n; ++i) {
    cout << "a[" << i
         << "] の分子を入れて下さい ";
    cin >> a[i].bunsi;
    cout << "a[" << i
         << "] の分母を入れて下さい ";
    cin >> a[i].bunbo;
  }
  // 実行前の状態の表示
  cout << "¥n¥n 並べ換える前 ¥n¥n";
  hyouji(a,n);
  // ソート
  for (made=n; made>0; --made) {
    for (k=1; k<made; ++k) {
      s1=(a[k-1].bunsi)*(a[k].bunbo);
      s2=(a[k-1].bunbo)*(a[k].bunsi);
      if (s1>s2)
        bswap(a[k-1],a[k]);
    }
  }
  // 出力
  cout << "¥n¥n 並べ換えた後 ¥n¥n";
  hyouji(a,n);
}
```

【実行例】

```
n=10
a[0] の分子を入れて下さい 1
a[0] の分母を入れて下さい 2
a[1] の分子を入れて下さい 1
a[1] の分母を入れて下さい 3
a[2] の分子を入れて下さい 2
a[2] の分母を入れて下さい 3
a[3] の分子を入れて下さい 1
a[3] の分母を入れて下さい 5
a[4] の分子を入れて下さい 2
a[4] の分母を入れて下さい 5
a[5] の分子を入れて下さい 3
a[5] の分母を入れて下さい 5
a[6] の分子を入れて下さい 4
a[6] の分母を入れて下さい 5
a[7] の分子を入れて下さい 1
a[7] の分母を入れて下さい 7
a[8] の分子を入れて下さい 3
a[8] の分母を入れて下さい 7
a[9] の分子を入れて下さい 5
a[9] の分母を入れて下さい 7

並べ換える前
  1   1   2   1   2   3   4   1   3   5
--- --- --- --- --- --- --- --- --- ---
  2   3   3   5   5   5   5   7   7   7

並べ換えた後
  1   1   1   2   3   1   3   2   5   4
--- --- --- --- --- --- --- --- --- ---
  7   5   3   5   7   2   5   3   7   5
```

【解説】 分数の大小は次のようにして判定しています．

$$\frac{a_{k-1}\text{ の分子}}{a_{k-1}\text{ の分母}} \quad と \quad \frac{a_k\text{ の分子}}{a_k\text{ の分母}}$$

を通分すると

$$\frac{(a_{k-1}\text{ の分子})\ (a_k\text{ の分母})}{(a_{k-1}\text{ の分母})\ (a_k\text{ の分母})} \qquad \frac{(a_{k-1}\text{ の分母})\ (a_k\text{ の分子})}{(a_{k-1}\text{ の分母})\ (a_k\text{ の分母})}$$

になりますから，その分子の方だけ

$(a_{k-1}$ の分子）$(a_k$ の分母）　　$(a_{k-1}$ の分母）$(a_k$ の分子）

を比較すればよいわけです．

関数 **bout2** は，n 個の分数を前のページの実行例のような形式で表示するためのプログラムで，

$n < 20$

分子，分母とも 3 桁以内

ということを前提にしています．

bout2 を呼び出すときにはデータを「構造体配列」の形で渡しますので，引数としては配列名を渡せばよく，配列名はポインタでもあるので **&** を付ける必要がありません．それに対し **bswap** を呼び出すときには，（配列でない）構造体を渡すので **&** を付ける必要があります．**a** の型宣言に ***** が付いているのは **new** で実体を生成したときポインタが渡されるからです．両者の違いに注意しながらプログラムを読んで下さい．

初心者はこの節をとばしてよい

6.5 共 用 体

いくつかの変数に同じ記憶場所を割り当てたいことがあります．そのための手段が共用体で，たとえば

```
// example 6.4a
#include <iostream.h>

union doukyo{    // 共用体の宣言
  float a;
  int   i;
};

main( )
{
  doukyo d;      // 実体の宣言

  cin  >> d.a;
  cout << d.i;
}
```

【実行例】

```
123.567
8782
```

と書けば，

実数型変数 **d.a**

整数型変数 **d.i**

に同じ記憶場所が割り当てられます．確かに同じ番地になっているかどうかは，& を付けてポインタを出してみればわかりますね．

```
// example 6.4b
#include <iostream.h>

main( )
{
  // 簡単な書き方の例
  union {
    float a;
    int   i;
  } d;

  cin  >> d.a;
  cout << d.i;
}
```

【実行例】

```
987.654
-5669
```

共用体の宣言は，普通，次の形で書きます．

```
union  共用体の名前 {
       型名  個別名 ;
       型名  個別名 ;
         …以下同様…
};
```

の形でデータの構成を書き，

```
union  共用体の名前  共用名 ;
```

によって，その変数が特定の形式の共用体であることを宣言します．

これを引用するときは必ず

共用名 . 個別名

の形で二つの名前を組にして用います．

(例)　前のページのように宣言した場合

```
j=i+1;
```

というような個別名単独の利用はできません．

人の名前の「姓」に相当するのが共用名，「名」に相当するのが個別名で，必ず両者を組合わせて使うわけです．なお，「型名」の所に

```
struct  {構造体の項目構成}
union   {共用体の内容構成}
```

などを書くことも許されています．

また，もう少し簡単な書き方もできます．

```
union {
        型名　個別名 ;
          …以下同様…
} 共用名 ;
```

これは，型の宣言と共用名の宣言を同時に行い，共用体の名前を省略したものです．

実は，構造体も同様に

```
struct {
        型名　項目名 ;
          …以下同様…
} 変数名 ;
```

と書けます．

struct と union は，同じように書けると覚えておくとよいでしょう．

プログラム例6.4——ダ・ディ・ダ

共用体に数値を代入し，一つの値を二つの違う型で出力してみましょう．

```
// example 6.4c
#include <iostream.h>

// a と i に同じ記憶場所を共用させる
  union yuming{
    float a;
    long   i;
  };

main( )
{
  yuming d;   // d は yuming 型の共用体
  cout << " 最初の状態 ¥n";
  cout << " d.a= " << d.a << '¥n';
  cout << " d.i= " << d.i << "¥n¥n";
  cout << " d.a に 3.14 を入れます ¥n";
  d.a=3.14;
  cout << " d.a= " << d.a << '¥n';
  cout << " d.i= " << d.i << "¥n¥n";
  cout << " d.i に 1993 を入れます ¥n";
  d.i=1993;
  cout << " d.a= " << d.a << '¥n';
  cout << " d.i= " << d.i << '¥n';
}
```

【実行例】

```
最初の状態
d.a= 6.22467e-012
d.i= 752550626

d.a に 3.14 を入れます
d.a= 3.14
d.i= 1078523331

d.i に 1993 を入れます
d.a= 2.79279e-042
d.i= 1993
```

【解説】 同じ番地に，最初，実数型の **7.7**，次に整数型の **123**，最後に実数型の **7.7** を代入したので，最終的に **7.7** が入っています．**d.a** を実数書式（%**f**）で出力すれば **7.7** が表示され，**d.i** を整数書式（%**d**）で出力すると，実数値 **7.7** のビット列を整数値として読んだ値が表示されます．

6.6　クラスの書き方

クラスの宣言は，普通，次のように書きます．

```
class クラスの名前 {
    まとめて扱うデータの型と名前を
        型名 変数名;
    の形式で列挙
public:
    一般利用者に使わせる関数を
        型名 関数名( 仮引数の列 ) {
            処理手続き
        }
    の形で記述して列挙
一般利用者に見せない関数があれば private: と書いて
    そのプログラムを同様な形式で列挙
}
```

実体（インスタンス）

クラスや構造体を使うとき，名前（識別名）が

クラスの名前　　（型名に相当する）
実体の名前　　　（変数名に相当する）

の２種類あることに注意して下さい．

たとえば，時，分，秒をまとめて「時間」という形で扱うクラスを定義し，そのクラスに `jikan` という名前を付けたとしましょう．それを使って，

出発した時刻を `a` とする
到着した時刻を `b` とする
経過時間（すなわち `a` と `b` の差）`c` を計算する

というプログラムを書くとすると，

`jikan`　　がクラスの名前
`a, b, c`　　がその実体

です．クラスの実体を宣言するには次のように書きます．

```
クラスの名前　その実体の名前を列挙 ;
（例） jikan a,b,c;
```

こうして宣言しておけば，あとは普通の変数と同様に，代入文の右辺，左辺に書いたり，関数呼出しの引数として書くことができ，後述の「演算子の定義」をしておけば，

```
c=b-a;
```

というように，式の形で用いることもできます．

メンバー

クラスは「データをまとめて扱うための手段」ですから，いうなれば「団体旅行者」であり，まとまって行動するのが原則です．でも，場合によっては，その中の個々のデータを呼び出したいこともあるでしょう．

クラスの中の個々のデータを**メンバー**といいます．サークルの部員だと思えば理解しやすいでしょう．正確にいうと，データだけではなく，クラス宣言の中に書いてある関数もメンバー（メンバー関数）と呼びます．部室にある道具も仲間の内と考えているのですね．

構造体の場合には，メンバーを表すのに

　　変数名 . メンバー名　　（例）`a.bunbo`

という書き方をしました．クラスでも同じように，

　　実体の名前 . メンバー名

で指示することができます．しかし，構造体とちょっと違って，クラスの場合には，一般利用者は原則としてメンバーに直接アクセスできません．この点については，次の項を見て下さい．

プライベートとパブリック

クラスは，普通，ブラックボックスとして使われます．これまで度々使ってきた `cin` や `cout` はその好例です．中の仕掛を知らなくても，またクラスという形で扱われていることさえ知らなくても，ブラックボックスとして使うことができるのです．逆に言えば，クラスとは「ブラックボックスを作るための手段」であり，その文法はブラックボックスを作るのに便利なように決められています．一般利用者がメンバーに直接アクセスできない，という規則もそのためで，原則として舞台裏には一般客を入れないようにしてあるのです．

その原則を曲げて，特に一般公開を許す場合には，

```
public:
```

と書きます．これを書いておけば，その後に書かれたメンバーは「クラスの外からアクセス可能」になります．これを書かなければ（あるいは明示的に

```
private:
```

と指定すれば）クラスの外から見えなくなります．

この規則は，メンバー変数だけでなく，メンバー関数についても適用されます．したがって，うっかり**public:**を書き忘れると，外でそれを使おうとしたとき，

　　そのような関数は見つかりません

というエラーメッセージが出るでしょう．

用語解説

クラスと実体　聞き慣れない用語なので，わかりにくいかも知れませんが，要するに

　　クラスは型（自分で定義できる型）

　　実体はその型の変数

だと思って使って下さい．実体という呼び方を避けて，変数名といって教えてもいいのですが，そうすると他の文献やマニュアルを読む時に混乱するので，本書では原語のままにしておきます．

メンバー　これは「クラスの構成要素」です．メンバー名というのは，カズとかラモスというような個人名ではなく，球技でいえばポジション名であり，「〜というチームのゴールキーパー」というような表し方をするわけです．

プログラム例6.5――時間差

時，分，秒を組にして扱う jikan というクラスを作り，その

入力　出力　時間差の計算

を行なう関数を作ってみましょう．

```
//  example 6.5a
#include <iostream.h>

// クラスの定義
class jikan {
  int h;         // 時
  int m;         // 分
  int s;         // 秒

  public:
  // キーボードから時間を読込む
  void in(void)
  {
    jikan aa;
    cout << " 時,分,秒を入れて下さい ";
    cin >> h >> m >> s;
  }

  // 時間の差を計算
  jikan sa(jikan aa,jikan bb)
  {
    jikan cc;
    int   ms=0,hm=0;
    // 秒の引き算
    if ( aa.s>=bb.s )
       cc.s=aa.s-bb.s;
    else {
       cc.s=aa.s+60-bb.s;
       ms=1;
    }
    // 分の引き算
    if ( aa.m-ms>=bb.m )
       cc.m=aa.m-ms-bb.m;
    else {
       cc.m=aa.m+60-ms-bb.m;
       hm=1;
    }
```

```
      // 時の引き算
         cc.h=aa.h-hm-bb.h;
      return cc;
   }

   // 表示
   void out(void)
   {
      cout << h << " 時間  " ;
      cout << m << " 分  " ;
      cout << s << " 秒  " << " ¥n ";
   }

};               // クラス宣言の終了

// メインプログラムの例
main( )
{
   jikan a,b,c;

   a.in();                      // a を読み込む
   b.in();                      // b を読み込む

   c=c.sa(a,b);                 // 差の計算

   cout << " a= " ; a.out();        // a を表示
   cout << " b= " ; b.out();        // b を表示
   cout << " c= " ; c.out();        // c を表示
}
```

【実行例】

```
時,分,秒を入れて下さい 11 22 33
時,分,秒を入れて下さい 10 20 30
a= 11 時間 22 分 33 秒
 b= 10 時間 20 分 30 秒
 c= 1 時間 2 分 3 秒
```

6.7 演算子を定義する方法

演算子の定義は次のように書きます.

```
クラス名 operator 記号 ( 仮引数 ) {

    計算手続きをプログラムの形で記述

}
```

(例) 前記のプログラムの jikan というクラスにマイナスの演算子を追加して, 時間 t1 と t2 の差を t1-t2 と表現できるようにするための演算子定義は次のように書きます.

```
 //  example 6.5b
#include <iostream.h>

class jikan {
  int h;          // 時
  int m;          // 分
  int s;          // 秒

  public:

  // キーボードから時間を読込む
  void in(void)
  {
    jikan aa;
    cout << " 時,分,秒を入れて下さい ";
    cin >> h >> m >> s;
  }

  // 整数型の h m,s を時間型にまとめる
  void atai(int hh,int mm,int ss)
  {
    h=hh;
    m=mm;
    s=ss;
  }
```

```
// 時間の和を計算
jikan wa(jikan aa,jikan bb)
{
  jikan cc;
  int   ms=0,hm=0;
  // 秒の和
  cc.s=aa.s+bb.s;
  if ( cc.s>=60 ) {
     cc.s-=60;
     ms=1;
  }
  // 分の和
  cc.m=aa.m+bb.m+ms;
  if ( cc.m>=60 ) {
     cc.m-=60;
     hm=1;
  }
  // 時の和
  cc.h=aa.h+bb.h+hm;
  return cc;
}

 // 時間の差を計算
jikan sa(jikan aa,jikan bb)
{
  jikan cc;
  int   ms=0,hm=0;
  // 秒の引き算
  if ( aa.s>=bb.s )
     cc.s=aa.s-bb.s;
  else {
     cc.s=aa.s+60-bb.s;
     ms=1;
  }
  // 分の引き算
  if ( aa.m-ms>=bb.m )
     cc.m=aa.m-ms-bb.m;
  else {
     cc.m=aa.m+60-ms-bb.m;
     hm=1;
  }
  // 時の引き算
  cc.h=aa.h-hm-bb.h;
  return cc;
}
```

```
    // 表示
    void out(void)
    {
      cout << h << " 時間 " ;
      cout << m << " 分 " ;
      cout << s << " 秒 " << '¥n';
    }

};              // クラス宣言の終了

// 演算子の定義
jikan operator +(jikan a,jikan b)
{
  jikan c;
  return c.wa(a,b);
}

jikan operator -(jikan a,jikan b)
{
  jikan c;
  return c.sa(a,b);
}

// メインプログラムの例
main( )
{
  jikan a,b,c,d;

  a.in();               // a を読み込む
  b.atai(1,2,3);        // 1時 2分 3秒 を b に入れる

  c=a+b;                // 和の計算
  d=a-b;                // 差の計算

  cout << " a= " ; a.out();      // a を表示
  cout << " b= " ; b.out();      // b を表示
  cout << " c=a+b ¥n";
  cout << " d=a-b ¥n";
  cout << " c= " ; c.out();      // c を表示
  cout << " d= " ; d.out();      // d を表示
}
```

【実行例】

```
時,分,秒を入れて下さい 11 22 33
a= 11 時間 22 分 33 秒
b= 1 時間 2 分 3 秒
c=a+b
d=a-b
c= 12 時間 24 分 36 秒
d= 10 時間 20 分 30 秒
```

プログラム例6.6――分数計算パッケージ

分数の入出力と加減乗除をサポートするクラス BNS を作ってみましょう.

```
// example 6.6
#include <iostream.h>

class BNS {
   long bunsi;
   long bunbo;

   public:
   // 初期化の方法を指定
   BNS(void) {
      bunsi=0;
      bunbo=1;
   }

   // 演算子+の定義
   friend BNS operator +(BNS& a,BNS& b)
   {
      BNS c;
      c.bunsi=(a.bunsi)*(b.bunbo)+
         (a.bunbo)*(b.bunsi);
      c.bunbo=(a.bunbo)*(b.bunbo);
      return c;
   }

   // 演算子-の定義
   friend BNS operator -(BNS a,BNS b)
   {
      BNS c;
      c.bunsi=(a.bunsi)*(b.bunbo)-
               (a.bunbo)*(b.bunsi);
      c.bunbo=(a.bunbo)*(b.bunbo);
      return c;
   }
```

```
// 演算子＊の定義
friend BNS operator *(BNS a,BNS b)
{
  BNS c;
  c.bunsi=(a.bunsi)*(b.bunsi);
  c.bunbo=(a.bunbo)*(b.bunbo);
  return c;
}

// 演算子／の定義
friend BNS operator /(BNS a,BNS b)
{
  BNS c;
  c.bunsi=(a.bunsi)*(b.bunbo);
  c.bunbo=(a.bunbo)*(b.bunsi);
  return c;
}

// 整数型の分母と分子から分数を作る
friend BNS bunsuu(int a,int b)
{
  BNS r;
  r.bunsi=a;
  r.bunbo=b;
  return r;
}

// 分数の読み込み
friend void read(BNS& a)
{
  cout << " 分子 ";
  cin >> a.bunsi;
  cout << " 分母 ";
  cin >> a.bunbo;
}
```

```
    // 分数の表示
    friend void write(BNS& a)
    {
      cout << a.bunsi ;
      cout << " / " ;
      cout << a.bunbo ;
      cout << "¥n";
    }
};   // 最後のセミコロンを忘れないこと！

main( )
{
    BNS a,b,wa,sa,seki,shou;

    cout << "分数 a を入れて下さい¥n";
    read(a);
    b=bunsuu(1,3);      // 1/3 を b に入れる

    wa=a+b;                // 和の計算
    sa=a-b;                // 差の計算
    seki=a*b;              // 積の計算
    shou=a/b;              // 商の計算

    cout << "a=";   write(a);
    cout << "b=";   write(b);
    cout << "a+b=";   write(wa);
    cout << "a-b=";   write(sa);
    cout << "a*b=";   write(seki);
    cout << "a/b=";   write(shou);
}
```

【実行例】

```
分数aの分子と分母を入れて下さい
1 2
a=1 / 2
b=1 / 3
a+b=5 / 6
a-b=1 / 6
a*b=1 / 6
a/b=3 / 2
```

付録 A

ポ イ ン タ

ここでは「CやC++を学習するときの最大の難所」と言われているポインタについて説明します．「難所」といっても，凡人に理解できないほどムズカシイものではなく，よく考えて読めば誰にもわかることなのですが，

常識の世界にない概念である
他の言語にも類例がない*
いいかげんに使うと失敗する

ので，注意深く読んで下さい．

なお，C++では，以前（Cの時代）ほどポインタを頻繁に使わなくても済むようになりましたから，初心者は必要になってから読めばいいと思います．

* PASCALのポインタはCのポインタとかなり違います．

A.1　番　地　の　話

ご存知のように，メモリ（記憶装置）には番地がついています．0番地から始まって，普通，数万番地まで使用できます．

番地は1バイト（8ビット，すなわち2進法の8桁で，文字を一つ格納できる広さ）単位についています．

短い整数型（short）	は	2バイト	整数型はこの
長い整数型（long）	は	4バイト	どちらか一方．
実数型（float）	は	4バイト	
倍長実数型（double）	は	8バイト	

で表されますので，記憶場所はそれを考慮して割り当てられます．

（例）

実数型変数 a	実数型変数 x	整数型変数 i	整数型変数 j	長い整数型変数 d	倍長実数型変数 c	6字の文字列*

機械語でプログラムを書くときは，データを取り出したり格納したりするとき，いつも番地で指定します．

「1000番地のデータを取り出しなさい」

「それに1を加えなさい」

「結果を1004番地に入れなさい」

というぐあいです．高級言語（BASIC, COBOL, FORTRANなど）の場合はコンピュータが自動的に番地を割り当てて計算してくれるので利用者は番地を意識しません．しかしCでは「番地を適当に使う」ことによって，効率を上げたり，キメ細かい処理をしたりできるようになっています．

---

* 終端記号が必要なので正味5字．

上級者のためのメモ

実際のコンピュータの番地のつけ方はもう少し複雑になっています．それは

- **多重処理**　何人もの人が同時に1台のコンピュータを使用します．その場合，プログラムは「現在，メモリの空いている場所」にロードして実行されますから，どこにロードされてもよいようにしておく必要があります．
- **仮想記憶**　実際のメモリ構成に関係なく非常に大きな仮想メモリ空間があると思ってプログラムを書いておくと，計算機が実情に合わせてヤリクリしながら実行してくれる方式です．
- **モジュール化**　プログラムをモジュール（部分品）単位に作成しそれを自由に組み合わせて使用することがよく行なわれます．その場合，やはり，どこにロードされても実行できるようにしておく必要があります．

などのためで，具体的には「仮の番地」を割り当てておいて，

リンクの際に調整
ロードの際に調整
実行時にベース・アドレスを加算

などの処理を経て最終的な番地が決まってきます．

そういうわけで，ポインタの値は「本当の番地そのもの」ではなくてある種の相対的な番地なのですが，本書では簡単に「番地」と呼んでおくことにします．

A.2 & 記号と ＊ 記号

CやC++では「変数の格納場所（番地)」を

& 変数名

で表し，「ある番地に格納されている変数の値（内容)」を

＊ 番地

で表します．いいかえれば

> x が変数名のとき
>
> &x
>
> は「x に割り当てられた番地」を表し，a が番地のとき
>
> ＊a
>
> は「a 番地の内容」を表す．

というわけです．

ただし，CやC++の文法では，これを「番地」といわないで「ポインタ」と呼びます．すなわち

> ＊a は 「ポインタ型変数 a が指す変数の値」

といいます．

プログラム例 A.1 ポインタを読んでみる

実際に「ポインタ」がどんな値になっていて，その指す先と，どういう関係になっているか，表示して調べてみましょう．

```
// example A.1
#include <iostream.h>

main( )
{
  int    n=753;
  float  h=3.1416;
  double k=123.4567;
  cout<<"&n = "<<int(&n)<<'¥n';
  cout<<"&h = "<<int(&h)<<'¥n';
  cout<<"&k = "<<int(&k)<<'¥n';
}
```

【実行例】

```
&n = 3912
&h = 3908
&k = 3900
```

プログラム例 A.2 ―― 2 + 3 = 5

ポインタを使って

2 + 3 = 5

の計算をやってみましょう．

```
// example A.2
#include <iostream.h>
#include <new.h>

main( )
{
  int* a=new int;     // a はポインタ
  int* b=new int;     // b はポインタ
  int* c=new int;     // c はポインタ
  *a=2;
  *b=3;
  *c=(*a)+(*b);
  cout << "*a=" << *a << '¥n';
  cout << "*b=" << *b << '¥n';
  cout << "*c=" << *c << '¥n';
}
```

【解説】 a, b, c を「整数型へのポインタ」と宣言し，その内容として *a には 2 を，*b には 3 を代入し，加算した結果を *c に代入しています．

【実行例】

```
*a=2
*b=3
*c=5
```

プログラム例 A.3 ―― *pc=*pa+*pb

今度は *pa, *pb の値をキーボードから読み込んで

    *pc=*pa+*pb;

の計算をしてみましょう.

```
// example A.3
#include <iostream.h>
#include <new.h>

main( )
{
  int* pa=new int;     // pa はポインタ
  int* pb=new int;     // pb はポインタ
  int* pc=new int;     // pc はポインタ
  cin >> *pa;
  cin >> *pb;
  *pc=(*pa)+(*pb);
  cout << "*pa=" << *pa << '¥n';
  cout << "*pb=" << *pb << '¥n';
  cout << "*pc=" << *pc << '¥n';
}
```

【解説】 new int というのはC++の新しい書き方です. こうするとポインタだけでなく記憶場所も確保されます.

【実行例】

```
123
456
*pa=123
*pb=456
*pc=579
```

A.3 配列名，文字列名はポインタ

CやC++の文法では

配列名 a はポインタである

その値は &a[0] すなわち配列の先頭番地に等しい

ということになっています．

文字列は「文字型の配列」ですから，文字列名もポインタであり，それは文字列の先頭を指しています．

（例）
```
// example A.4
#include <iostream.h>

main( )
{
  char a[20];
  cout<<"a    ="<<int(a)<<'¥n';
  cout<<"&a[0]="<<int(&a[0])<<'¥n';
  cout<<"&a[1]="<<int(&a[1])<<'¥n';
  cout<<"&a[2]="<<int(&a[2])<<'¥n';
}
```

を実行すると次のようになります

```
a    =3206
&a[0]=3206
&a[1]=3207
&a[2]=3208
```

2 次元配列の場合も，第 2 添字を省くとポインタになります．

(例) わかり易いように文字型配列で調べてみましょう．

```
// example A.5
#include <iostream.h>

main( )
{
  char c[10][12];
  int i;
  for (i=0; i<10; ++i) {
    cout<<"c["<<i<<"]=";
    cout<<int(c[i])<<'¥n';
  }
}
```

【実行例】

```
c[0]=3090
c[1]=3102
c[2]=3114
c[3]=3126
c[4]=3138
c[5]=3150
c[6]=3162
c[7]=3174
c[8]=3186
c[9]=3198
```

	$j=0$	$j=1$	$j=2$			$j=11$
$i=0$	-166	-165	-164	…	…	-155
$i=1$	-154	-153	-152	…	…	-143
$i=2$	-142	-141	-140	…	…	-131
$i=3$	-130	-129	-128	…	…	-119
$i=4$	-118	-117	-116	…	…	-107
$i=5$	-106	-105	-104	…	…	-95
$i=6$	-94	-93	-92	…	…	-83
$i=7$	-82	-81	-80	…	…	-71
$i=8$	-70	-69	-68	…	…	-59
$i=9$	-58	-57	-56	…	…	-47

C_{ij} の格納場所一覧表

A.4 ポインタに1を加えると…

ポインタは番地ですから整数の値をとります．それに1を加えたら，いくつになるでしょうか？「1を加えれば1ふえる」と思うでしょう？ところが，さにあらず．実際は2ふえたり，4ふえたりするのです．実際に，やってみましょう．

```
// example A.6
#include <iostream.h>

main( )
{
  float *a;
  cout<<"a   = "<<int(a   )<<'¥n';
  cout<<"a+1= "<<int(a+1)<<'¥n';
  cout<<"a+2= "<<int(a+2)<<'¥n';
  cout<<"a+3= "<<int(a+3)<<'¥n';
}
```

【実行例】

```
a   = 11483
a+1= 11487
a+2= 11491
a+3= 11495
```

ほら，四つずつふえるでしょう? これは

`float *a;`

によって「aは実数型へのポインタである」と宣言したからで，もし

`int *a;`

と宣言すれば（普通は）二つずつふえます．実数型はデータ1個が4バイトであり，整数型は（普通）2バイトだからです．一般に

`*(a+i)` は `a[i]` と同じ

になるようにしてあるのです．

A.5　ポインタ配列

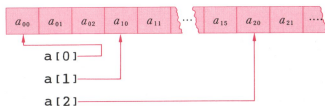
最後に

`float *a[10];`　または　`float* a[10];`

というような宣言の意味についてご説明しておきましょう．* 印が付いていますから `a[i]` はポインタです．添字が付いていますから「ポインタの表」すなわち「ポインタ配列」です．

ポインタ配列の使いみちはいろいろありますが，特に重要なのは「1行ごとに長さの違う表」です．

a_{00}	a_{01}	a_{02}				
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	
a_{20}	a_{21}	a_{22}	a_{23}			
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}
a_{40}						
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}		
a_{60}	a_{61}					

これをコンパクトに記憶させるため，データを1列に並べて，「各行の先頭番地」をポインタ配列に入れておくのです．

他の言語でも，これと同じようなことはできます．しかし，このように格納された個々の配列要素を a[i][j] という普通の配列と全く同じ形で使うことのできるのはCだけです．これは

　　a[i]　は　*(a+i)　を表す

　　a[i][j]　は　*(a[i]+j)　を表す

というC独特の文法のおかげなのです．

プログラム例 A.4――メモリの動的割当ての例

ご参考までに，入力した元数 n に合わせてメモリを確保して連立1次方程式を消去法で解くプログラムを示しておきます．

```
#include <iostream.h>
#include <malloc.h>
#define N 100

int main()                    // メインプログラムの例
{
  int   i,j,n,nn,nbt,nnbt;
  float *a[N+1];       // 係数行列
  float *b;            // 定数項
  float *x;            // 解
  float s;             // 作業場所
  void leq(float* a[],float b[],
             float x[],int n);
  // メモリー割り付け
  cout << " 元数 n を入れて下さい ¥n";
  cin >> n;
  nn=n+1;
  if (nn>N) {
    cout << "memory over ¥n";
    return (1);
  }
  nbt=4*nn;
  nnbt=4*nn*nn;
  a[0]=(float *)malloc(nnbt);
```

```
    b=(float *)malloc(nbt);
    x=(float *)malloc(nbt);
    for (i=1; i<nn; ++i) {
      a[i]=a[i-1]+nn;
    }
    // テスト・データ生成
    for (i=1; i<=n; ++i) {
      s=0.0;
      for (j=1 ;j<=n ;++j) {
        a[i][j]=(i<j) ? i:j;
        s+=a[i][j]/(float)j;
      }
      b[i]=s;
    };
    // 計算のプログラムを呼び出す
    leq(a,b,x,n);
    // 結果の表示
    for (i=1; i<=n; ++i) {
      cout << x[i] << '¥n';
    }
    cout << "終了";
    return (0);
}

// 連立1次方程式を解く
void leq(float* a[],float b[],
              float x[],int n)
{
    int i,j,k;
    float p,q,s;
    cout << "計算開始¥n";
    for (k=1 ;k<n ;++k) {
      p=a[k][k];
      for (j=k+1; j<=n; ++j) {
        a[k][j]=a[k][j]/p;
      }
      b[k]=b[k]/p ;
      for (i=k+1; i<=n; ++i) {
        q=a[i][k] ;
        for (j=k+1; j<=n; ++j) {
          a[i][j]=a[i][j]-q*a[k][j];
        }
        b[i]=b[i]-q*b[k] ;
      };
    };
```

```
    x[n]=b[n]/a[n][n];
    for (k=n-1; k>=1; --k) {
      s=0.0 ;
      for (j=k+1; j<=n; ++j) {
        s=a[k][j]*x[j]+s;
      }
      x[k]=b[k]-s;
    } ;
}
```

演　習　問　題

2 章

2.1　次の言葉を画面に表示するプログラムを作ってみましょう．

(1)　Happy Birthday

(2)　ヤッター

(3)　合格祈願

ヒント　パソコンのC++ではたいてい文字列定数（" "印で囲まれた中）に漢字を使うことができます．PC98系のマシンならば，" 印を打鍵してから CTRL キーを押しながら XFER キーを押すと漢字を入力できるモードになります．漢字入力が終わったら，再び CTRL ＋ XFER 操作をして，漢字入力モードを解除し，それから " キーを打鍵します（漢字入力モードのままで " キーを押すと全角の " 印が入力され，それはC++の記号になりませんのでエラーになってしまいます）．

2.2　上問のプログラムに，注釈（たとえば

`exercise 2.1`　　エンシュウ 2.1　　演習 2.1

など）を付けてみましょう．

ヒント　パソコンのC++ではたいてい，注釈に漢字を使うことができます．

2.3　自分の住所，氏名を宛名シールの形式で表示するプログラムを作ってみましょう．

(例)　埼玉県　与野市　鈴谷

　　　1993-12

　　　名 草 千 枝 子 様

2.4　二つの整数 m, n を読み込み，$m \div n$ の計算をして結果を表示するプログラムを作ってみましょう．また，そのプログラムを実行させて，

$3 \div 2$　　$2 \div 3$　　$2 \div 0$　　$0 \div 0$

のとき，どんな答が出るか実験してみましょう．

2.5　整数 a, b, c, d を入力し，次の計算をしてそれぞれの結果を表示するプログラムを作ってみましょう．

(1)　$2a + 3b$　　(2)　$(a + b)(a - b)$

(3)　$ab - cd$　　(4)　$2(abc - d)$

(5)　$\frac{b}{a} + \frac{b}{c}$　　(6)　$\frac{c + d}{a + b}$

2.6 次の計算のプログラムを作ってみましょう．その際，変数名の付け方に工夫して，自然で読み易いプログラムになるよう努力して下さい．

(1) 人口密度 = 人口 ÷ 面積

(2) 抵抗 = 電圧 ÷ 電流

(3) 金額 = 数量 × 単価

2.7 h_1 時 m_1 分 s_1 秒から h_2 時 m_2 分 s_2 秒までの（経過）時間を計算するプログラムを作ってみましょう．

ヒント 時分秒で表されている時間をそのまま引き算しようとすると，60進法の繰り上がり（繰り下がり）の処理が必要になって煩雑になるので，それを避けるため，秒単位に直して計算するのが賢明です．その結果を時分秒表現に戻すには，まず3600で割って h を求め，その余りを60で割って m を求め，その余りを s にします．余りは46ページで説明した % 演算子で計算します．

2.8 インクリメント演算子 ++ の効果を調べるために，

```
i=1;    cout<<i<<'¥t';
cout<<i++<<'¥t'<<i++<<'¥n';
i=1;    cout<<i<<'¥t';
cout<<++i<<'¥t'<<++i<<'¥n';
i=10;   cout<<i<<'¥t';
cout<<i--<<'¥t'<<i--<<'¥n';
i=10;   cout<<i<<'¥t';
cout<<--i<<'¥t'<<--i<<'¥n';
cout<<(i*=2)<<'¥n';
cout<<(i*=2)<<'¥n';
```

というような内容のプログラムを作って実行させてみましょう．

2.9 次の単位の換算をするプログラムを作ってみましょう．

(1) インチ → cm　　cm → インチ　　(長さ)

(2) 平米 → 坪　　坪 → 平米　　(面積)

(3) 摂氏 → 華氏　　華氏 → 摂氏　　(温度)

2.10 次の計算のプログラムを作ってみましょう．

(1) 円の面積を計算する

(2) 台形の面積を計算する

(3) ヘロンの公式で三角形の面積を計算する

ヒント x の平方根は sqrt(x) で計算できます（59ページ参照）．

2.11 多項式

$ax^3 + bx^2 + cx + d$

の値を計算するプログラムを作ってみましょう.

ヒント 上の式は次のように変形できます.

$((ax + b)\,x + c) + d$

CやC++は累乗 x^n の計算が書きにくく，計算時間もかかるので，多項式の計算はこのように変形してプログラムに書くのが定石です.

2.12 58ページで説明した関数を使って

$\sin 30°$　　$\cos 30°$　　$\sin 45°$

などの値を計算し，理論値

$1/2$　　$\sqrt{3/2}$　　$\sqrt{2/2}$

と比較する（差を表示する）プログラムを作ってみましょう.

2.13 直交座標 X, Y の値を読込み，極座標 r, θ に変換して表示するプログラムと，逆に r, θ を入力し，X, Y に変換して表示するプログラムを作ってみましょう.

ヒント 計算式は次のようになります（ただし θ の符号に要注意).

$r = \sqrt{X^2 + Y^2}$　　$X = r\cos\theta$

$\theta = \arctan Y/X$　　$Y = r\sin\theta$

2.14 整数 n を読み込み，

$y = (1 + 1/n)^n$

の値を計算して出力するプログラムを作り，

$n = 10$　　$n = 100$　　$n = 1000$

などを入れて計算させてみましょう.

ヒント 59ページで説明した関数 `pow` を使います．引数の型の取扱いをよく考えて下さい.

2.15 2次方程式

$a^2x + bx + c = 0$

の係数 a, b, c を読み込み，その根を計算して表示するプログラムを作ってみましょう．ただし，2章までの段階では判別式の符号を調べて計算式を切換えることができませんので，実根だけを扱うことにします.

2.16 整数 n を読込み，シフト演算子 `<<`, `>>` による

```
n<<1   n<<2   n<<3
n>>1   n>>2   n>>3
```

などの結果を表示するプログラムを作って実行させてみましょう．n に負の数を入れた場合はどうなるでしょうか？

3 章

3.1　入力促進メッセージとして

「あなたの身長を入れて下さい（単位は cm）」

を表示して身長を読込み，もし 180 cm 以上であれば

「ずいぶん背が高いのですね」

と表示するプログラムを作ってみましょう．

3.2　実数値 x を読込み，

$|x| \leqq 1$　ならば　$y = \sqrt{1 - x^2}$

$|x| > 1$　ならば　$y = 0$

の値を計算して表示するプログラムを作ってみましょう．

3.3　整数 n を読込み，偶数か奇数かを調べて，結果を表示するプログラムを作ってみましょう．

ヒント　46ページで説明した % という演算子を使って「n を 2 で割った余り」を求め，それが 0 ならば偶数，1 ならば奇数と判定します．

3.4　1000万円をローンで借りて，毎年100万円ずつ返済することにします．年利率が 5 %として，完済までの毎年のローン残高を計算して表示するプログラムを作ってみましょう．

3.5　毎年10万円ずつ，元利合計が300万円になるまで積立てることにします．毎年の元利合計を計算し表示するプログラムを作ってみましょう．ただし，年利率は 3 %で，利息の20%は税金で引かれるものとします．

3.6　百万円を年金の原資とし，1 年後から毎年10万円ずつ年金を支給する場合の，各年度の年金支給後の残高を残高が10万円未満になるまで計算して表示するプログラムを作ってみましょう．なお年利は 3 %とし，利息の20%は税金で引かれるものとします．

3.7　実数値 x を読込み，

$a_0 = x \qquad b_0 = 1/x$

から出発して，

$k = 0, 1, 2, 3, \cdots$

の順に，

$a_{k+1} = (a_k + b_k)/2$

$b_{k+1} = 1/a_{k+1}$

という規則で

$|a_k - b_k| < 0.0001$

になるまで，数列

$a_0, a_1, a_2, a_3, \cdots a_k$

$b_0, b_1, b_2, b_3, \cdots b_k$

を計算するプログラムを作ってみましょう（この数列は，どちらも，$\sqrt{x}$ に収束することが知られています）.

3.8 1から50までの整数の

逆数　平方　平方根

の数表を出力するプログラムを作ってみましょう.

3.9 $n=1$ から $n=16$ までの 2^n およびその逆数 (2^{-n}) の数表を作ってみましょう.

3.10 次の級数の第10項までの和を計算して表示するプログラムを作ってみましょう.

(1) $1+\dfrac{x}{1!}+\dfrac{x^2}{2!}+\dfrac{x^3}{3!}+\cdots+\dfrac{x^k}{k!}+\cdots$

(2) $1-\dfrac{x}{1!}+\dfrac{x^3}{3!}-\dfrac{x^5}{5!}+\cdots+\dfrac{(-1)^k x^{2k-1}}{(2k-1)!}+\cdots$

ヒント $x^k/k!$ に $x/(k+1)$ を掛ければ $x^{k+1}/(k+1)!$ になります．(2)は2項ずつまとめて計算するとよいでしょう.

3.11 整数 n を読込み，その約数を全部求めて表示するプログラムを作ってみましょう.

ヒント 1から n までの全部の整数で割ってみて，余りが0ならば約数ですから表示します.

3.12 1000までのすべての素数を求めて表示するプログラムを作ってみましょう.

ヒント 前問と同じように，その数以下の全部の整数で割ってみるのが最も簡単です．しかし，そのままでは能率がよくありませんので，いろいろ工夫して下さい．それとは全く違う方法として，エラトステネスの篩（ふるい）というアルゴリズムもあります.

3.13 九九の表を作るプログラムを書いてみましょう.

ヒント i を1から9まで変え，その中で j を1から9まで変えて，積 ij を（きれいな表の形で）出力すればよいわけです．出力桁数をそろえるには **¥t**（タブ機能）を使うのが1案ですが，**setw**（桁数）という書き方を使う方法もあります（103ページ参照）.

3.14 カレンダーを作るプログラムを書いてみましょう.

ヒント まず，当月1日が日曜（または月曜）から始まる場合を考え（作ってみて）それに成功したら，「1日の曜日をキーボードから入れる（たとえば，月

曜ならば1，火曜なら2，…とすれば処理し易い)」という方式を試みるとよいでしょう．より一般的なプログラムの作り方については，たとえば拙著「PC9800シリーズRA/RS/RX/ES/EX BASICとその応用」(サイエンス社) などを参考にして下さい．

3.15　小学生のための，計算練習のプログラムを作ります．

(1)　iとjを適当に決め (たとえば$i=2, j=3$ならば)

```
2+3=
```

というような形式で画面に問題を表示し，キーボードから入力された「答」をコンピュータで計算した$i+j$の結果と比較し，

　合っていれば「よくできました」

　違っていれば「残念でした」

を表示するプログラムを作ってみましょう．

(2)　10題の問題をやらせて，その内の正解の個数，まちがえの個数を計数して結果を表示するように改造してみましょう．

(3)　switch 文を使って，

　足し算　　引き算　　掛け算　　割り算

の練習を選べるように機能追加してみましょう．

ヒント　i, jを作るには，乱数生成のプログラムを使うのが理想的です．その方法は処理系によって違いますが，マイクロソフトC++の場合には

```
#include <Stdlib.h>
```

を冒頭に置き，

```
i=rand()%100
j=rand()%100
```

というようなぐあいに書くことができます (このように100で割った余りをとれば2桁の乱数が得られます)．

3.16　自分で乱数を作る方法としては，合同法と呼ばれている次のようなやりかたがあります．

準備　乱数のタネN_0を適当に決めます (たとえば1993)．

生成　前の乱数N_kをもとに，新しいN_{k+1}を

$$N_{k+1} = (N_k \text{を} a \text{倍して} m \text{で割った余り})$$

で作ります．そのプログラムを作ってみましょう．

aとmの選定方法については深い数学的な研究があります (Knuth 著，渋谷

政昭訳，準数値算法/乱数，サイエンス社）．しかし適当にデタラメな数を使ってやってみて，うまくいかなければ取り替える，という方法でも，前問のような目的には使えます．ただし，m の桁数が少ないとすぐ循環してしまいますので，m はなるべく大きくとって，まず桁数の多い乱数を作り，実際に使うときには前記のようにもう一度 % をして桁数の少ない乱数に直します．実際には「m で割る」という手間を省くため「整数 a を掛けたときのオーバーフローを無視する」のが普通で，これによって，$m = 2^{32}$ で割ったのと同じ効果が得られます．そのような m を用いる場合の a の推奨値としては，1664525とか69069などがKnuth の本に紹介されています．

4 章

4.1 二つの 3 次元ベクトル

$$\boldsymbol{a} = (a_1, a_2, a_3)$$

$$\boldsymbol{b} = (b_1, b_2, b_3)$$

の和

$$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$$

を計算するプログラムを作ってみましょう．

4.2 二つの 3 次元ベクトル $\boldsymbol{a}, \boldsymbol{b}$ の内積（スカラー積）

$$(\boldsymbol{a}, \boldsymbol{b}) = a_1b_1 + a_2b_2 + a_3b_3$$

を計算するプログラムを作ってみましょう．

4.3 二つの 3 行 3 列の行列

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

の和

$$A + B = C = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

を計算するプログラムを作ってみましょう．

4.4 行列とベクトルの積

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

を計算するプログラムを作ってみましょう．

4.5 二つの正方行列の積

$$AB = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

を計算し，結果を配列 C に入れるプログラムを作ってみましょう．

4.6 3行3列の単位行列を作って配列 A に入れるプログラムを作ってみましょう．

ヒント 単位行列の定義は，$i = j$ ならば $a_{ij} = 1, i \neq j$ ならば $a_{ij} = 0$

4.7 3次の行列式

$$\det A = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

の値を計算するプログラムを作ってみましょう．

4.8 三角形の頂点の座標

(x_1, y_1)　　(x_2, y_2)　　(x_3, y_3)

が与えられたとき，面積を計算するプログラムを作ってみましょう．

ヒント 面積 S は行列式

$$\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}$$

の絶対値を2で割ることによって求められます．

4.9 n 人の生徒の

身長　x_1　x_2　x_3　$\cdots$　x_{n-1}　x_n

体重　y_1　y_2　y_3　$\cdots$　y_{n-1}　y_n

を読み込み，

身長の平均値 μ_x　　身長の標準偏差 σ_x

体重の平均値 μ_y　　体重の標準偏差 σ_y

身長と体重の相関係数 ρ

を計算するプログラムを作ってみましょう．

ヒント ρ は次の式で計算できます．

$$\rho = \sum_{i=0}^{n} \frac{(x - \mu_x)(y - \mu_y)}{n\sigma_x\sigma_y}$$

4.10 キーボードから文字列を読込み，それを3行並べて表示するプログラムを作ってみましょう．

(例)「春は曙，秋は小錦」と入力したら

春は曙，秋は小錦
春は曙，秋は小錦
春は曙，秋は小錦

と表示するわけです．

4.11　キーボードから名前を読込み，それを歌に詠込んで，

Happy birthday to you!
Happy birthday to you!
Happy birthday（ここに名前を挿入）san,
Happy birthday to you!

と表示するプログラムを作ってみましょう．

4.12　文章の中の TOYOTA という綴りを全部 NISSAN に置換えるプログラムを作ってみましょう．

4.13　文字列を一つ読込み，それが「前から読んでも後から読んでも同じ綴り」(逆順にしても変わらない綴り）であるかどうか調べるプログラムを作ってみましょう．

(テスト・データの例)　SOS　12321　キツツキ

4.14　英語の文章を読込み，その中に ing という綴りが何箇所あるか調べるプログラムを作ってみましょう．

ヒント　関数 `strcmp` を使って，

まず先頭の 3 字と比較
次の 3 字と比較

というように文末まで調べるのが 1 案ですが，別案としては，まず文字 i を探し，i を見つけたら「その次は n か？」「その次は g か？」と照合していく方法も考えられます．

4.15　いくつかの単語を読込み，文字数の少ない順に並べ換えて表示するプログラムを作ってみましょう．

ヒント　文字列の長さは `strlen` という関数を使って得ることができます．

(例)　文字列 `a` の長さは `strlen(a)` です．

注意　`strlen` を使う際には

```
#include <string.h>
```

が必要です．

4.16　いくつかの名前を読込み，その中に

e で終る名前

ko で終る名前

mi で終る名前

がそれぞれ幾つあるか数えるプログラムを作ってみましょう.

5 章

5.1 実数型の引数 x を受取り，その 2 乗 x^2 を返す関数 `nijou(x)` を作ってみましょう.

5.2 実数型の引数 x を受取り，

$x < 0$ ならば $f(x) = 0$

$x \geqq 0$ ならば $f(x) = 1$

を実数型で返す関数 $f(x)$ を作ってみましょう.

5.3 整数型の引数 n を受取り，その逆数 $1/n$ を返す倍精度実数型の関数 `gyakusu(x)` を作ってみましょう.

ヒント 整数型のまま割り算すると（$n \geqq 2$ のとき）答が 0 になってしまいますから，倍精度実数型への変換は割り算を行なう前に済ませておく必要があります.

5.4 整数型の引数 n を受取り，

$1 + 2 + 3 + \cdots + n$

を計算して返す関数 `madenowa(n)` を作ってみましょう.

ヒント for 文を使ってマトモに計算することもできますが，

$S = n(n-1)/2$

という公式で計算しても構いません.

5.5 引数としてデータの個数 n と，一組のデータ

$a_1, a_2, \cdots, a_n$

を受取り，その平方和

$S = a_1^2 + a_2^2 + \cdots + a_n^2$

を計算して返す関数 `heihouwa(a,n)` を作ってみましょう.

5.6 2 項係数（n 個の内から r 個をとる組合わせの数）

${}_nC_r = n!/(r!(n-r)!)$

を計算するプログラムを作ってみましょう. 引数は n と r で整数型. 結果は関数値として整数型で返します.

ヒント $n!$ の計算にはプログラム例 5.11 を使用できますが，for 文を使って 1 から n までの整数の積を計算しても簡単です.

5.7 整数 n を受取り，

偶数ならば１（真）　奇数ならば０（偽）

を返す関数 `guusuu(n)` を作ってみましょう．

5.8 ある関数の数表

$y_k = f(x_k) \quad (k = 0, 1, \cdots, n-1)$

が配列 x と配列 y に格納されています．実数値 x が引数として与えられたとき，この数表を補間して

$y = f(x)$

の値を計算し，関数値として返すプログラムを作ってみましょう．

ヒント x が与えられたら，まず

$x_k \leqq x < x_{k+1}$

になるような k を探します．それが見つかったら，

$y = y_k + (x - x_k)(y_{k+1} - y_k)/(x_{k+1} - x_k)$

によって y を計算します（これは最も簡単な「線形補間」の公式です．より精密な補間の公式に関しては数値計算の文献を見て下さい）．

ヒント 配列 x, y およびそのデータ数 n を大域変数として扱えば，関数を

```
double f(double x)
```

の形式で作ることができます．

5.9 行列 A, B の積 C を計算する関数

```
gs(a,b,c,l,m,n)
```

を作ってみましょう．ただし，a, b, c は行列 A, B, C の配列名で，

A は m 行 l 列　B は l 行 n 列　C は m 行 n 列

とします．

5.10 n 人の生徒の期末試験の点数（素点）

$s_1, s_2, s_3, \cdots, s_{n-1}, s_n$

が整数型の配列 `s[]` に入っています．これに60点のゲタをはかせるため，

$t_i = 60 + 0.4 s_i \quad (i = 1, 2, \cdots, n)$

という処理をする関数

```
geta60(s,t,n)
```

を作ってみましょう．

5.11 前問のプログラムを改造して，変換後の最低点が50点，最高点が100点になるようにしてみましょう．また，平均点が75点，標準偏差が10点になるようなプログラムも考えてみましょう（厳密にいうと，これは少々むずかしいのです．100点以上を認めれば簡単ですが，100点に抑えると平均値，標準偏差が変わってしまうので再調整が必要になります）．

5.12　身体検査のデータ

学生番号	1	2	3	…	$n-1$	n
身長	x_1	x_2	x_3	…	x_{n-1}	x_n
体重	y_1	y_2	y_3	…	y_{n-1}	y_n

があります．これを身長の大きい順に並べ換えるプログラムを作ってみましょう（学生番号や体重も身長のデータに合わせて並べ換えて下さい）．

5.13　10進法表現の整数が文字列の形で与えられたとき，それを整数型の形に変換する関数 **atoi(s)** を作ってみましょう．

ヒント　文字列 **s** の先頭に符号（**+-**）があればその種別を記憶しておきます．部分和 **n** に最初は 0 を入れておきます．次に，文字列 **s**（ただし符号は省く）の先頭から数字 $\mathtt{m_i}$ を一つずつ取出し，

$\mathtt{n+=10*n+m_i}$

を文字列の最後まで繰返し，最後に符号を付けます．

5.14　与えられた文字列 **s** の中に，アルファベットの各文字がそれぞれ何回使われているか数えて，結果を整数型配列 **h** に格納する関数を作ってみましょう．

ヒント　文字データ（1字だけ）は 0〜255の整数で表されているので，寸法255の整数型配列を用意し，文字を添字に使ってカウントしていくことができます．

5.15　フィボナッチ数列の第 n 項を求める関数 $f(n)$ を，再帰呼出しの形で書いてみましょう．

5.16　n 次の行列式の値を計算するプログラムを作ってみましょう．

ヒント　拙著「BASIC による線形代数」（共立出版）を参考にして工夫して下さい）．

6 章

6.1　実数部と虚数部を組にして複素数を扱うための構造体 **complex** を定義し，その

入力　　出力

加算　　減算　　乗算　　除算

絶対値をとる　　偏角を求める

などのプログラムを書いてみましょう．

6.2　平面上の座標 x, y をまとめて扱うための構造体を定義し，その

入力（x と y を読み込む）

出力（x と y を表示する）

平行移動（移動量 ξ, η を与えて計算）

回転（原点を中心に θ だけ回転）

のプログラムを書いてみましょう．

ヒント 回転は，行列の乗算

$$\begin{bmatrix} 新x \\ 新y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

によって計算できます．

6.3 3次元ベクトルを扱う構造体を定義し，その

入力　　出力

和　　差　　スカラー倍

内積（スカラー積）　外積（ベクトル積）

を計算する関数を作ってみましょう．

6.4 名前を

漢字　　カタカナ　　ローマ字

の組で扱う構造体を定義し，

(1) 五十音順に並べる

(2) アルファベット順に並べる

のプログラムを作ってみましょう．

ヒント 厳密な意味での五十音順の並べ換えはかなり高等技術に属します．ここでは一応，JIS コードの順にソートすればよいことにしましょう．

6.5 名前を

姓　　名　　（いずれもローマ字）

の組で扱う構造体を定義し，

(1) 姓の ABC 順に並べ換える

(2) 名の ABC 順に並べ換える

のプログラムを作ってみましょう．

6.6 ある学級の生徒の名簿が，前問で定義した構造体の配列の形で表されているとします．そのとき，同姓同名があるかどうかを調べるプログラムを作ってみましょう．

6.7 日付を

年（整数型）　月（整数型）　日（整数型）

の組で扱う構造体を定義し，年月日 `d1` から年月日 `d2` までの日数を計算する関数 `nissuu(d1,d2)` を作ってみましょう．

ヒント 月の日数が一定でなく，さらに閏年があるため，かなりやっかいな計算になります．いろいろな方法が考えられますが，最も簡単なのは，日付をまず固

定した時点（たとえば1900年1月1日）からの日数に換算し，その引き算をする，という方法です．このような計算のやりかたに関しては拙著「PC9800シリーズRA/RS/RX/ES/EX BASICとその応用」（サイエンス社パソコンライブラリ）にプログラム例があるので参考にするとよいでしょう．

6.8　多項式

$$a_nx^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_2x^2 + a_1x^1 + a_0$$

を，整数 n と配列 `a[]` の組として扱うクラス `polyno` を定義し，

多項式の和　　多項式の差　　多項式の積

x の値を代入して多項式の値を求める

微分する　　積分する

などのプログラムをメンバー関数の形で作ってみましょう．

6.9　数ベクトルを，

名前（文字列）

次元（整数型）

要素（倍精度実数型配列）

の組として扱うクラスを定義し，

和　差　内積　スカラー倍

などのプログラムを作ってみましょう．

6.10　行列を

名前（文字列）

行数，列数（整数型）

要素（倍精度実数型2次元配列）

の組として扱うクラスを定義し，

和　差　積　入力　出力

などのプログラムを作ってみましょう．

6.11　前問の和差積を演算子 `+-*` で書けるようにしてみましょう．

6.12　待行列を扱うためのクラスを作ってみましょう．

ヒント　メンバー関数として，

`in`　待行列に入る（新しいオブジェクトを列の最後に追加する）

`out`　待行列を出る（先頭のオブジェクトを取り出して順につめる）

を用意します．

6.13　スタック扱うためのクラスを作ってみましょう．

ヒント　メンバー関数として，

`push`　スタックに入れる

pop　　スタックから取り出す

を用意します．処理は待行列に似ていますが，

　　push のとき，新しいオブジェクトを列の最後に追加する

　　pop　のとき，列の最後のオブジェクトを取り出す

という点が違います．

6.14　辞書を管理するクラスを作り，メンバー関数として

　　新しい単語を辞書に追加する

　　辞書を引く

を作ってみましょう．話を簡単にするため，

　　dog　　イヌ

　　cat　　ネコ

というように，単語が1対1に対応するものとします．

6.15　子供銀行の普通預金口座を管理するクラスを作り，

　　預ける　　引き出す　　残高照会

などのためのメンバー関数を作ってみましょう．

6.16　木構造（二分木）を扱うクラスを作ってみましょう．

ヒント　これは少々むずかしいので，わからなくなったらC++の専門書を参考にして下さい．カーニハン/リッチの「プログラミング言語C」（石田晴久訳，共立出版）には構造体による実現方法が書かれています．

索　　引

あ　行

か　行

さ　行

た　行

な　行

記　号

著 者 略 歴

戸川隼人（とがわ はやと）

1935年　東京に生まれる
1958年　早稲田大学第一理工学部数学科卒業
科学技術庁航空宇宙技術研究所，
京都産業大学，日本大学理工学部を経て，
現　在　尚美学園大学教授　理学博士

主 要 著 書

マトリクスの数値計算
微分方程式の数値計算
有限要素法による振動解析
有限要素法へのガイド
数値解析とシミュレーション
共役勾配法
ザ・C
演習と応用 FORTRAN77
UNIX ワークステーションによる
科学技術計算ハンドブック［基礎篇 C 言語版］

NS ライブラリ＝5
ザ・C++

1993年12月25日 ©　　初 版 発 行
2006年2月10日　　初版第10刷発行

著　者　戸 川 隼 人　　発行者　森 平 勇 三
印刷者　小宮山一雄

発行所　**株式会社　サイエンス社**
〒151-0051　東京都渋谷区千駄ヶ谷1丁目3番25号
営　業　☎(03)5474-8500(代)　振替00170-7-2387
編　集　☎(03)5474-8600(代)
FAX　☎(03)5474-8900

印刷・製本　小宮山印刷工業㈱
《検印省略》

ISBN 4-7819-0718-0

PRINTED IN JAPAN

サイエンス社のホームページのご案内
http://www.saiensu.co.jp
ご意見・ご要望は
rikei@saiensu.co.jp　まで．

サイエンス社のホームページのご案内
http://www.saiensu.co.jp
ご意見・ご要望は　rikei@saiensu.co.jp　まで.

ISBN4-7819-0718-0 C3341 ¥1900E　　定価(本体1900円+税)